AFRL-IF-RS-TR-2005-222
Final Technical Report
June 2005

SURVIVABILITY USING CONTROLLED SECURITY SERVICES

USC Information Sciences Institute

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. H545/J029

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2005-222 has been reviewed and is approved for publication

APPROVED: /s/
JON B. VALENTE
Project Engineer

FOR THE DIRECTOR: /s/
WARREN H. DEBANY, JR.
Technical Advisor
Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 2005	**Final** **Aug 99 – Sep 03**

4. TITLE AND SUBTITLE

SURVIVABILITY USING CONTROLLED SECURITY SERVICES

5. FUNDING NUMBERS

G - F30602-99-1-0530
PE - 62301E
PR - H545
TA - 10
WU - 01

6. AUTHOR(S)

B. Clifford Neuman, Joseph Bannister, Dan Boneh, Gene Tsudik

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

USC Information Sciences Institute
4676 Admiralty Way
Marina Del Rey CA 90292-6695

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington VA 22203-1714

AFRL/IFGA
525 Brooks Road
Rome NY 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

AFRL-IF-RS-TR-2005-222

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Jon B. Valente/IFGA/(315) 330-1559 Jon.Valente@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

This document represents the final report on the SUCSES project funded by DARPA/ITO under the "Dynamic Coalitions" (DC) program. SUCSES' period of performance spanned roughly four years between August 1999 and September 2003. The document is organized as follows. First part contains the overview of the project's setting and its goals. We then present a summary of year-by-year technical achievements and activities. Next, we list all external results including: publications, presentations, dissertations and technology transfer efforts. The bulk of the report is contained in the appendix that includes the copies of all publications produced under the aegis of the SUCSES project.

14. SUBJECT TERMS

Survivability, cryptography, digital signatures, privacy, identify-based encryption

15. NUMBER OF PAGES 275

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Tables

Introduction and Overview

Many modern distributed services, network protocols and applications are deployed over insecure and unreliable public networks, such as the global Internet. Communication security is based upon availability of timely, efficient and effective security services. Some of the most formidable security challenges are represented by the high cost of security and by the insufficiently controlled access to security services. To this end, this project proposed two new directions in information security. The first applied to dynamic networks where security capabilities of end-users must be tightly controlled. The second applied to networks containing low-powered devices such as PDAs and smartcards. Our approach uses partially trusted security mediators and is well suited for both military and civilian applications. Following are the highlights of the work:

- Tightly Controlled Security Services. Many situations call for a security officer to quickly revoke, reinstate or alter an end-user's ability to take part in critical security tasks. We say that a capability is tightly controlled if it can be quickly revoked at will. Our architecture provides tight control over the following end-user capabilities: (1) generation of digital signatures, (2) decryption of messages, and (3) authentication to others. We achieve this without relying on Certificate Revocation Lists (CRLs) and other off-line means. Instant, fine-grained control of security services is particularly useful in battle-field scenarios. (For example, all security capabilities of captured equipment or personnel must be immediately disabled.) It is also well-suited to civilian applications where activities—such as signing contracts and decrypting sensitive data—must be both revocable and tightly controlled.

  We obtain tight control over security capabilities by requiring each end-user to interact with an untrusted SEcurity Mediator (SEM). An end-user cannot generate a valid signature or decrypt a message without the SEM's help. To revoke an end-user's security capability, the security officer simply instructs the SEM to refuse any interaction with the said end-user thereby attaining instant revocation. The mediator is carefully designed to minimize both interaction and computation so as to increase robustness and availability. In the same vein, SEM is physically distributed to facilitate scalability, fault tolerance and minimize risks from denial-of-service attacks.

- Networks of Weak Devices. In networks containing low-powered devices (e.g., PDA-s) security operations often represent a performance bottleneck. As an outgrowth of the above, we offload heavy security computations to an untrusted SEM. We aim to do so without compromising end-device's secrets and by taking advantage of the superior computing power of SEM.

  - Aided key generation. Generating cryptographic keys on weak devices such as PDAs and cell-phones takes on the order of many minutes: both the delay and power consumption are of concern. We developed techniques that enable a weak device to offload much of the key generation work to an un-trusted SEM. This way, key generation delay is reduced to a few seconds. At the end of the computation, the SEM has no information about the secret key it helped generate. This kind of aided key generation can be used by smartcards as well as other embedded devices.

  - Aided signature computation. Similar to the above, we investigated and developed methods for offloading costly components of public key signature generation to untrusted mediators. We considered several methods for doing so. The end-device needs to compute only a few hashes per signature while retaining full benefits of public key-based signatures.

  - We demonstrated the effectiveness of our approach by implementing all new techniques as a library of mediated cryptographic services. The library implements both tight control mechanisms and off-loading of security computations. The usage of wrappers enabled easy integration of our code into existing projects/products. This work resulted in numerous insights into the delicate and difficult integration of cryptographic protocols with practical communication systems.

- Identity Based Encryption. The complexity of managing public-key certificates motivated Adi Shamir to propose the idea of Identity Based Encryption (IBE) back in 1984. Unfortunately no usable implementations of IBE were known. During the course of this project, and motivated by the

complexity of revoking certificates, we discovered a very practical IBE system. An IBE system is a public-key encryption mechanism where public keys can be arbitrary strings, such as email addresses, phone numbers, or dates. At a high level, IBE essentially eliminates the need for certificates since a person's public-key is just that person's public identity. During the course of the project we successfully demonstrated that IBE leads to simple public-key management. We implemented an IBE library and integrated IBE with several mail readers. In 2002 several of our students launched a start-up company, Voltage Security, to commercialize IBE. The company participated in this year's JWID and the resulting reviews are very favorable.

Project Accomplishments: Year 1

As the initial step in the project, an extensive literature review has been undertaken in order to locate any current or past research efforts with goals similar to those of SUCSES. A number of activities were identified.

The project Web page was designed and advertised in several security-related conferences and Internet newsgroups. It includes, among other things, project goals, problem statement, quad charts as well as other documents and publications.

Starting in mid-August we began working on the initial architecture of the SUCSES toolkit. Basic layering design has been done; however, a low-level modular arithmetic package has not been selected yet. Since there were a number of public domain software packages, selecting the optimal one required some experimentation with all.

The SAS component of SUCSES has undergone several revisions. The most recent version has the following features:

- Support for X.509 certificates
- Seamless migration of SEMs
- Fully-configurable interface and system parameters
- MOTIF-based user interface
- Signed error messages from SEM
- Platform independence (SUNos, SOLARIS and LINUX)
- Documented API

The following activities were started during the first year of the project and completed in later years:

- SAS plug-in for Netscape Communicator
- Measurements for SAS versus plain-RSA overhead
- Integration (unification) with Stanford's communication platform

One of our main goals was to off-load expensive part of generating RSA keys on a handheld device such as the PalmPilot. Our prototype showed how to speed up RSA key generation by a factor of six without compromising the security of the generated key. We obtained this speedup by offloading most of the key generation work onto an untrusted server. The server did most of the work, but learned nothing about the key it helped generate.

During the past year we extended our RSA server aided architecture to also support RSA signature generation. The resulting prototype can be used for both RSA key generation and RSA signature generation. We use the PalmPilot as an example handheld device. We chose the PalmPilot since it is easy to program and to experiment with.

When using 1024 bit RSA keys our new prototype implementation shows that RSA signature generation can be sped up by a factor of 2.5. We used a 500 MHz Pentium III as the server. The implemented

protocol is based on new practical techniques for generating RSA signatures with the help of an untrusted server. Both results on RSA key generation and RSA signatures were reported in the technical literature.

Our primary research result this year is the novel scheme for mediated (SEM-assisted) RSA signatures and RSA decryption. Mediated RSA (MRSA) is different from SAS in that the actual MRSA signature is indistinguishable from a plain RSA signature. The approach can be roughly outlined as follows:

Setup:

1. CA generates a number of half-RSA keys, one for each SEM.
2. CA assigns to each client (user) is assigned a half-RSA key as well.

Signature:

1. When client requests an RSA signature, he sends the text to SEM
2. If client is not revoked, SEM computes a half-signature with its key and returns the result
3. Client proceeds to use his half-key to obtain an actual RSA signature (which he also verifies!)

Mediated decryption works in a similar manner.

The interesting security properties are: 1) no communication between CA and SEM; 2) Computational overhead only double of plain RSA; 3) Verification unchanged from plain RSA; 4) No certificates for SEMs (we use id-based assignment of half-keys; see below); 5) No per-user state on the SEM; 6) SEM knows no user secrets, cannot impersonate users;

The initial (proof-of-concept) implementation of MRSA is taking place over the summer. We conducted extensive measurements of its effectiveness and costs.

One important goal of the SUCSES project is to show how a SEcurity Mediator (SEM) can be used for key management and key revocation. During this first year we added another feature to the SEM design: identity-based encryption.

Identity-based encryption considerably simplifies certificate management. Currently, when Alice wants to send a message to Bob she must first obtain Bob's public key certificate. When using identity-based encryption, Bob's public key is simply Bob's Email address. Hence, Alice need not obtain Bob's certificate. She simply encrypts using Bob's Email address as a public key. Bob obtains the private key corresponding to his Email address from the CA. Unfortunately, there are yet no usable public key systems that support identity-based encryption. However, as is turns out, RSA can be easily converted into an identity-based system by using an untrusted SEM. All encrypted Email flows through the SEM. The SEM gains no information about the contents of the encrypted Email it is handling. As encrypted Email flows through the SEM, the SEM applies a certain simple operation to the Email header. The recipient then decrypts the message using his private key.

Project Accomplishments: Year 2

We briefly recall the objective of the SUCSES project: to develop novel technologies for survivable security services. The main goal is to provide organizations (military and civilian) with tight control—as opposed to today's certificate-based loose control—over end-users' security privileges such as digital signatures and data encryption/decryption. At the same time, SUCSES aims to greatly simplify public key certificate management by developing novel techniques that do not require the use of certificates. An additional goal is to assist weak computing devices (e.g., PDAs) by providing them with the same full security functionality (digital signatures, public key encryption, key generation) that is available for more powerful devices while retaining the ability to instantly revoke a device's access to security services.

The centerpiece of the SUCSES project is a component called Security Mediator (SEM). SEM is an on-line entity charged with assisting end-users (and end-devices) in obtaining security services. By mediating access to critical security services, a SEM acts both as a helper and an on-line revocation authority. Also, in the process of assisting end-users (specifically weak devices) SEM off-loads from them some of the

computational and storage burden involved in costly cryptographic operations. At the same time, an SEM is not entrusted with end-users' secrets and cannot impersonate and/or cheat constituent end-users without detection.

Using SEM-based mediation requires careful division of labor between the traditional role of an off-line Certification Authority (CA) and the on-line SEMs. Interactions between SEMs and end-users are kept to a minimum so as to make the SEM as unobtrusive as possible. Protocols between SEM and end-user are designed to be both provably secure and fault-tolerant.

Generating strong cryptographic keys on small devices such as a PalmPilot takes considerable time (tens of minutes). SUCSES is developing techniques to enable such devices to offload much of the public key generation work to an untrusted SEM. However, at the end of the computation, SEM has no information about the secret key it helped to generate. This way, key generation delay can be reduced to a few seconds. Aided key generation of this type can be used by smartcards as well as other embedded devices. In the same vein, investigation and development of methods for offloading costly components of public key signature generation to untrusted mediators will be undertaken. Several methods will be considered. Ideally, the end-device will need to compute only a few inexpensive hashes per signature while retaining full benefits of public key-based signatures.

ACCOMPLISHMENTS:

Most public key methods are based on a binding between users and their respective keys. Such bindings are usually expressed in certificates; this requires complex means of issuance, distribution and revocation. In contrast, identity-based encryption considerably simplifies certificate management. Currently, when Alice wants to send a message to Bob she must first obtain Bob's public key certificate. With identity-based encryption, Bob's public key is simply Bob's Email address. Hence, Alice need not obtain Bob's certificate. She simply encrypts using Bob's Email address as a public key. Bob obtains the private key corresponding to his Email address from the CA.

During the past year a new identity-based encryption scheme (IBE) has been developed. It is based on the concept of "Weil Pairing" on elliptic curves. Unlike SAS and mRSA, IBE requires no on-line interaction. IBE greatly simplifies public key certificate management since, in it, a user's public key can be derived from a unique identifier, such as an email address. IBE is not only a practical scheme, it also addresses what has been, for a long time, an important open problem in cryptography. IBE can also be used in order to perform secure delegation of cryptographic keys to untrusted or partially trusted devices.

The SAS, mRSA and IBE components of SUCSES have been fully implemented as public domain libraries and email plug-ins. The following software has been developed in the past year (all components are available for download from the SUCSES web page):

- mRSA and SAS client libraries
- IBE libraries
- IBE plug-in for Eudora
- SAS signature verification program for Netscape and Outlook
- SAS plug-in for Eudora
- mRSA plug-in for Eudora (Netscape and Outlook require no changes to verify mRSA signatures)
- Stand-alone Unix-based SEM daemon supporting both SAS and mRSA
- A collection of CA utilities for certificate management

Extensive performance tests of the software have been undertaken. The results clearly indicate that, for small devices, SAS performs significantly better than plain RSA when a SEM is located on the same LAN or campus network. (The performance of SAS degrades as the client-SEM distance grows.) mRSA tests demonstrate performance that is, on the average, 3-4 times worse than plain RSA. However, for interactive applications (such as email), the slowdown is not noticeable.

Experiments using an unoptimized version of the IBE system shows that encryption/decryption take about 0.3 seconds on an 800 MHz PIII. Hence, even an unoptimized version of the system is quite practical. We are currently optimizing the IBE library and hope to reduce the time for encryption/decryption to under 0.1

seconds. Other projects are welcome to use the implementation of the IBE system. The IBE library is open source.

Planned Technology Transition

Technical transfer discussions with Rainbow Technologies, ENTRUST, Microsoft and Verisign were held throughout the past year. Some of these contacts are still on-going, while others (Entrust and Rainbow) terminated because of the worsened economy in early 2001. Both Verisign and Microsoft were happy to learn about the capabilities of mRSA and its application to code signing. Last year, Verisign made a costly blunder when it issued a Microsoft code signing certificate to an unknown person. This mishap clearly illustrated the difficulty of revocation with current methods. As a result, both Verisign and Microsoft are interested in fast revocation technologies for code signing certificates. mRSA provides easy revocation compatible with today's deployed software.

The intent is to deploy both IBE and mRSA/SAS on the global Internet. To this end, a "secure" web site is being built to store the IBE root keys. Initially, this site will be hosted in a physically secured location at Stanford campus. As mentioned above, full deployment is expected to take place before the end of the year

Project Accomplishments: Year 3

We developed an open-source library implementing the new Identity Based Encryption scheme (IBE). Most of the effort went into optimizing the library for performance. Encryption and decryption now take 40ms on a 1GhZ Pentium III. The library is open source and is available for download at http://crypto.stanford.edu/ibe

We also prototyped an identity-based email system. This is part of a larger project for building an identity-based PKI. The email system supports Outlook, Eudora, Hotmail, and Yahoo mail. The system was deployed in year 3 and currently has approximately 200 regular users world wide.

The same crypto technology that made the new IBE system possible can also be used to provide very short digital signatures. These signatures are half the size of DSS signatures and provide the same level of security. Our short signature scheme has been implemented and is available for download along with the library implementing IBE. An interesting property of the signature scheme is that its performance is the reverse of the performance of RSA signatures: signature generation is fast where as signature verification is slow (signature generation is three times faster than RSA signature generation and the signature is about one tenth the size) Hence, this signature scheme is better suited than RSA for handheld devices that generate signatures.

An ID-based variant of mRSA (IB-mRSA) has been designed and implemented. IB-mRSA is a simple and practical identity-based signature and encryption method. Its main difference from mRSA is the use of identities in computing users' public keys. It is fully compatible with mRSA and plain RSA. Its performance (for data encryption and verification of signatures) is only slightly lower than that of mRSA. It offers a realistic transition scenario for gradual introduction of public key cryptography. Moreover, it combines the reduced reliance on certificates (common to all ID-based schemes) with the fine-grained revocation intrinsic to mRSA.

A forward-secure variant of mRSA (FS-mRSA) has also been designed and implemented. FS-mRSA provides extra security over mRSA by periodically evolving the user's private key. Consequently, even if the adversary compromises the user's secrets, he is unable to forge user's signatures rooted in the past. Furthermore, FS-mRSA supports the notion of "expiring encryption" whereby data encrypted for the user can be made valid up to a certain time. Thereafter, even the legitimate user cannot decrypt the data. Also, the adversary who eavesdrops on encrypted data and later compromises the user's secrets is unable to decrypt pre-recorded data. Finally, the added forward security feature in FS-mRSA does not degrade the performance (i.e., it is as efficient as normal mRSA).

The currently available software includes the following components (note that all components are available for download from the SUCSES web page):

- mRSA, SAS, FS-mRSA and IB-mRSA client libraries
- Microsoft Outlook and Eudora plug-ins for mRSA, FS-mRSA and IB-mRSA signatures
- Decryption helpers for mRSA, FS-mRSA and IB-mRSA encrypted messages (work withe MS Outlook, Eudora and Netscape Messenger)
- A stand-alone email proxy (instead of plug-ins) supporting SAS and mRSA signatures
- Stand-alone Unix-based SEM daemon supporting both SAS and mRSA
- A collection of CA utilities for certificate management
- Optimized IBE crypto library including support for short signatures
- Microsoft Outlook, Hotmail, Yahoo mail, Eudora, and Pine plug-ins for the IBE email system
- Key generation and user management system to be used with the IBE mail plug-ins

Technical contacts are still being maintained with Microsoft and Verisign regarding possible technology transfer venues.

Project Accomplishments: Year 4

One of the main research contributions of SUCSES has been the invention of a practical identity-based public key cryptosystem (IBE). IBE allows us, among other things, to build an Email system where a party A can send secure Email to party B, even if the latter has no public key yet. The IBE system is implemented and accessible via an easy-to-use plug-in for common mail readers such as Outlook and pine. The resulting system makes it very simple for people to communicate securely via email. A technical paper describing the identity-based encryption system (based on Weil Pairing) was presented in Crypto 2001.

The IBE implementation is now fully optimized: it takes about 30ms for encryption & decryption on a 1-GHz Pentium. Hence the system is fast enough to replace RSA in commercial applications. The implementation is open source and is available on the project's web site.

We completed a mail gateway for Windows for performing IBE e-mail encryption on the fly. The mail gateway enables us to encrypt/decrypt any e-mail sent from any Windows mail reader. The resulting system is a brand new approach for dealing with secure e-mail. We are hoping it will spread through the population in a viral-like effect (Alice sends secure mail to Bob causing Bob to send secure mail to Carol and so on).

We also deployed IBE email for world-wide use using Internet Explorer plug-ins for hotmail and Yahoo! mail. The Private Key Generator (PKG) is running at Stanford. We are currently tracking the adoption rate.

IBE-Related Results

The techniques we used to construct our IBE cryptosystem can be used to solve many open problems in cryptography. We give three examples that we believe are the most exciting (the second application is new for this reporting period).

Short digital signatures. Using the principles underlying our IBE system (the Weil pairing) we were able to construct a signature scheme where the signatures are very short. Signatures in our scheme provide the same security as DSS signatures, but are half the size. Such short signatures are important for bandwidth constrained applications and especially applications where a human is asked to type in a digital signature. This often comes up in software product activation mechanisms where a user types in a digital signature to unlock the software product.

Aggregate signatures. Surprisingly, the short signature scheme described above has some unexpected properties – it supports signature aggregation. Signature aggregation enables anyone to compress a list of signatures on different messages generate by many users into a single signature. This is useful, for example, for shortening certificate chains. Recall that a certificate chain of depth n contains n signatures

issued by each of the n Certificate Authorities. Aggregate signatures enable all these signatures to be aggregated into a single signature, thus shortening the certificate chain. Aggregate signature can also be used to reduce the bandwidth in secure routing protocols such as SBGP.

Public key searching on encrypted data. Suppose Alice sends Bob an encrypted email containing the keyword "Urgent". Bob would like his mail gateway to forward such email to Bob's pager. Unfortunately, with encrypted email the gateway cannot tell that the email contains the word "Urgent". Using our IBE mechanism we are able to build an email system in which Bob can give the mail gateway a key that enables the gateway to test whether the email contains the "Urgent" keyword, but the gateway learns nothing else about the email. An observer, who does not have the key, learns absolutely nothing about the email contents (other than the length of the email).

Other Accomplishments

In prior reports, we have described the SAS, mRSA and ID-mRSA components of the SUCSES project. We have completed the implementation (client libraries, SEM support, CA utilities and Eudora/Outlook plug-ins) of SAS, mRSA and ID-mRSA functions. During this reporting period, our development efforts concentrated mainly on developing installation packages and documentation for both MS Outlook™ and Eudora™ plug-ins.

The work on mediated Diffie-Hellman key exchange (mDH) is still in progress. Subsequent plans include integration of mDH with the popular Kerberos Authentication Service. In addition, we are working on a version of mRSA that supports so-called blind signatures. Such signatures are useful in environments where the SEM must be prevented from identifying particular messages signed on behalf of a client. Note that this does not contradict the revocation functionality of a SEM: a SEM can identify the requesting client and perform a real-time revocation check. However, a SEM is unable to (later) associate a signature with a particular signature request.

We have begun limited experimentation on the UCI campus with the above software components. Practical experience gained from using our implementation will be valuable in identifying usability issues and comparing our revocation approach with the traditional CRL-based techniques.

Technology Transfer

Voltage Security, a 2-year old start-up, is commercializing the IBE system developed in the course of the project. The company participated in this year's JWID and received very favorable reviews. The Voltage product provides backend identity management and frontend encrypted email and encrypted files. There is also an available IBE toolkit that companies can use to incorporate IBE into their products. Recently, GemPlus released an implementation of our IBE system on a GemPlus smartcard. The combined SAS/mRSA/IB-mRSA is being phased in for use at UCI ICS Department.

Conclusion

The SUCSES project has made fundamental contributions to the science and engineering of secure communications in a packet-switched network such as the Internet. Three broad areas of research were addressed in the project: control of security services, security in networks of low-performance devices, and identity-based encryption.

The project developed strong theoretical and practical results in the fine-grain control of security services, especially in group signatures and immediate revocation. The work resulted in a comprehensive study of how to implement safe, fast revocation of certificates and security capabilities on large networks. The architecture developed and demonstrated relies on the SEM security mediator, a distributed and highly scalable approach to tight security control. It has been shown to be resilient and highly robust against distributed denial-of-service attacks.

The project was one of the first to recognize the need for fresh approaches to enabling security in networks of marginally capable devices, such as cell phones and personal digital assistants. The problems of efficiently generating RSA keys on low-power handheld devices were highlighted and innovatively solved

in the project. This subfield has gone on to become a very active area of research in security. Of particular interest is the technique aiding weaker devices by allowing a more-powerful server to assist in the processing of information used in key generation and signature computation.

Finally, the pioneering work in identity-based encryption, where public keys can take on human-friendly formats (e.g. string names) rather than machine-oriented formats (e.g. abstract numbers). Building on suggestions made two decades ago but never pursued, the project developed, validated, and demonstrated actual IBE concepts that were incorporated in a prototype and picked up for commercialization by Voltage Security, Inc. The IBE system is operational in several installations, providing effective, easy-to-use public-key services to a growing community of users.

The SUCSES project has cut a wide swath during its period of performance, producing significant findings of broad applicability in privacy and security of networked and distributed systems. The results have seeded research in other related areas and have been adopted by the market for real products.

Project Personnel

Name	Title	Period of performance
Gene Tsudik	Project Leader ISI & UCI	throughout
Joseph Bannister	Project Leader USC/ISI	Sept 01 -- Sept 03
Dan Boneh	Project Leader Stanford	throughout
Xuhua Ding	Graduate Researcher	Sept 99 -- Sept 03
Paolo Montini	Visiting Student	Jan 00 -- July 00
Kemal Bicakci	Graduate Researcher	Sept 99 -- July 00
Ben Lynn	Graduate Researcher	Sept 00 -- Sept 03
Hovav Shacham	Graduate Researcher	Sept 00 -- Sept 03

Table 1: Project Personnel

Publications

Following are all project-related publications that appeared at refereed conferences/workshops and archival-quality journals:

1. Leak-free Group Signatures with Immediate Revocation, X. Ding and G. Tsudik and S. Xu, 24th International Conference on Distributed Computing Systems (ICDCS'04), 2004
2. Fine-grained Control of Security Capabilities, D. Boneh and X. Ding and G. Tsudik , ACM Transactions on Internet Technology, 2004
3. Simple Identity-Based Encryption with Mediated RSA, X. Ding and G. Tsudik, RSA Conference, Cryptographer's Track, 2003
4. Identity-based Mediated RSA, D. Boneh, X. Ding, G. Tsudik, International Workshop on Information and Security Applications (WISA), 2002
5. Experimenting with Server-Aided Signatures, X. Ding, G. Tsudik and D. Mazzocchi, Network and Distributed System Security Symposium, 2002
6. A Method for Fast Revocation of Public Key Certificates and Security Capabilities, D. Boneh, X. Ding, G. Tsudik and M. Wong, 10th Usenix Security Symposium, Washington D.C., 2001
7. Generating RSA Keys on a Handheld Using an Untrusted Server, N. Modadugu, D. Boneh, and M. Kim, RSA Conference, Cryptographer's Track, 2000
8. Server-Supported Signatures, N. Asokan, G. Tsudik and M. Waidner, Journal of Computer Security, November 1997

9. How to construct optimal one-time signatures, K. Bicakci, G. Tsudik and B. Tung, Computer Networks (Elsevier), Vol 43 (3), pp. 339-349, October 2003
10. On constructing optimal one-time signatures, K. Bicakci, G. Tsudik and B. Tung, 15th International Symposium on Computer and Information Sciences, October 2000
11. Weak Forward Security in Mediated RSA, G. Tsudik, 2002 Security in Computer Networks Conference (SCN'02), September 2002
12. Short signatures from the Weil pairing, D. Boneh**,** H. Shacham, and B. Lynn, J. of Cryptology, Vol. 17, No. 4, pp. 297-319, 2004
    Extended abstract in proceedings of Asiacrypt '01, LNCS Vol. 2248, Springer-Verlag, pp. 514-532, 2001
13. Identity based encryption from the Weil pairing, D. Boneh and M. Franklin, SIAM J. of Computing, Vol. 32, No. 3, pp. 586-615, 2003.
    Extended abstract in proceedings of Crypto '2001, Lecture Notes in Computer Science, Vol. 2139, Springer-Verlag, pp. 213-229, 2001
14. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps, D. Boneh, C. Gentry, H. Shacham, and B. Lynn, proceedings of Eurocrypt 2003, LNCS 2656, pp. 416-432, 2003
15. Short Signatures Without Random Oracles, D. Boneh and X. Boyen, proceedings of Eurocrypt 2004, LNCS 3027, pp. 56-73, 2004
16. Public key encryption with keyword search, D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, proceedings of Eurocrypt 2004, LNCS 3027, pp. 506-522, 2004
17. A Secure Signature Scheme from Bilinear Maps, D. Boneh, Ilya Mironov and Victor Shoup, proceedings of RSA-CT '03

Dissertations and theses:

- Xuhua Ding, Fine-Grained Control to Security Services, Ph.D. Thesis, Computer Science Department, University of Southern California, December 2003

In addition, a number of presentations were made throughout the course of the project. Most are listed below:

1. **Invited Keynote:** Survivability Using Controlled Security Services, Workshop on Information Security Applications (WISA'01), Seoul (Korea), September 2001
2. **Invited Talk:** Survivable Security, Joint US/EU Information Survivability Workshop, Malvern, UK, June 2000
3. **Invited Distinguished Lecture:** Controlling Access to Security Services, Harvey Mudd College, January 2001
4. **JHU-ISI Distinguished Lecture Series:** Fast Revocation of Credentials, Johns Hopkins University, October 2001
5. **UMd-ISR Distinguished Lecture:** SUCSES: Survivability Using Controlled Security Services, University of Maryland, College Park, October 2001
6. **Invited Talk**: Fast Credentials Revocation, Electronics and Telecom Research Institute (ETRI) Korea, September 2001
7. **Invited Talk:** Identity-Based Cryptography, PKNU Busan (Korea), August 2002
8. **DTC Guest Lecture:** Mediated Cryptography, University of Minnesota, February 2003
9. **Invited Talk:** RSA Conference, Feb. 2003

Awards

- At the DC PI meeting held in San Antonio in January 2002, the PIs Dan Boneh and Gene Tsudik were recognized for their research in the context of the SUCSES project with **DARPA Dynamic Coalitions Program Award for Excellence in Academic Research.**
- Dan Boneh and Matt Franklin received the **InfoWorld Innovators award** on May 24th, 2004 for the development of the Boneh-Franklin IBE system.

Software

The software developed in the course of the SUCSES project is (and always has been) publicly available from the SUCSES web site: http://sconce.ics.uci.edu/sucses as well as from the companion site at Stanford: http://crtypto.stanford.edu.

Deployment and Technology Transfer

Voltage Security, a 2-year old start-up, is commercializing the IBE system developed in the course of the project. The company participated in this year's JWID and received very favorable reviews. The Voltage product provides backend identity management and frontend encrypted email and encrypted files. There is also an available IBE toolkit that companies can use to incorporate IBE into their products. Recently, GemPlus released an implementation of our IBE system on a GemPlus smartcard

The SEM toolkit is (or has been) in use at NAI Labs, UC Irvine, Dartmouth, UIUC, SRI, USC/ISI and Stanford.

Leak-free Group Signatures with Immediate Revocation

Xuhua Ding*
School of Information Systems
Singapore Management University
xhding@smu.edu.sg

Gene Tsudik
Department of Computer Science
University of California, Irvine
gts@ics.uci.edu

Shouhuai Xu†
Department of Computer Science
University of Texas at San Antonio
shxu@cs.utsa.edu

Abstract

Group signatures are an interesting and appealing cryptographic construct with many promising potential applications. This work is motivated by attractive features of group signatures, particularly, their potential to serve as foundation for anonymous credential systems. We re-examine the entire notion of group signatures from a systems *perspective and identify two new security requirements:* leak-freedom *and* immediate-revocation*, which are crucial for a large class of applications. We then present a new group signature scheme that achieves all identified properties. Our scheme is based on the so-called* systems architecture approach*. It is more efficient than the state-of-the-art and facilitates easy implementation. Moreover, it reflects the well-known* separation-of-duty *principle. Another benefit of our scheme is the obviated reliance on underlying anonymous communication channels, which are necessary in previous schemes.*

1. Introduction

The concept of group signatures was introduced by Chaum and van Heyst [18] in 1991. The chief motivation was to facilitate anonymous group-oriented applications. Given a valid group signature, any verifier can determine the signer's group membership, while only a designated entity (called a group manager) can identify the actual signer. In recent years, many research efforts sought efficient constructs and precise definitions of group signatures. Early schemes (e.g., [19]) suffered from linear complexities of either (or both) group public key size or signature size with respect to the number of group members. These drawbacks were first addressed by Camenisch and Stadler [13]. Follow-on results [12, 1] gradually improve on efficiency. Despite these advances, membership revocation remained a difficult obstacle [7, 33, 2]. Significant progress was made by Camenisch and Lysianskaya [9] who utilized a novel dynamic accumulator to construct a group signature scheme with reasonably efficient revocation. Further improvements have been made recently by Tsudik and Xu [34].

In the past, group signature schemes aimed to satisfy a jumble of (often redundant and overlapping) security requirements: *unforgeability*, *exculpability*, *traceability*, *coalition-resistance*, *no-framing*, *anonymity*, and *unlinkability* [13, 8, 12, 29, 3, 9]. To untangle and simplify these messy requirements, Bellare et al. [4] investigated "minimal" security requirements for group signature schemes. This line of research is very important, as it parallels the pursuit of similar requirements for *secure* public key encryption schemes [26, 31, 32, 22] and *secure* key exchange protocols. The work in [4] showed that two security properties: *full-traceability* and *full-anonymity* are sufficient to subsume all aforementioned (seven) requirements.

In this paper, we argue that *full-traceability* and *full-anonymity* are insufficient for certain realistic group signature settings. This claim is based on the observation that group signatures are inherently application-oriented, and, thus, cannot simply be treated as a primitive in the bare model. We identify two new requirements: leak-freedom and immediate-revocation which are necessary for a large class of commercial applications. Informally, leak-freedom means that a signer cannot

* Work done at the University of California, Irvine.
† Work done, in part, at the University of California, Irvine.

convince anyone[1] that it generated a given group signature. Whereas, immediate-revocation means that a group member's revocation results in immediate inability of generating group signatures, i.e., a revoked member cannot any further group signatures.

Why is leak-freedom **important?** One of the main purposes of group signatures is for an organization to conceal its internal structure. Suppose Alice is an employee of a company (say, ABC) authorized to sign purchase orders and one of the suppliers is another company XYZ. If Alice – without revealing her private key or any other long-term secret – can convince XYZ that she is the signer of a given purchase order, she could obtain illegal kickbacks from XYZ as "gratitude" for her supplier selection. This sort of *information leakage* illustrates potential abuses of group signatures, and justifies the introduction of the leak-freedom property to block such abuses.

Why is immediate-revocation **important?** We continue with the previous example: any purchase order signed by Alice (on behalf of her employer ABC) for a supplier XYZ imposes certain financial and/or legal responsibilities on ABC. However, if Alice is aware of her impending lay-off, she could, in collusion with XYZ, sign fraudulent purchase orders for ABC. (Note that this problem is less grave in traditional public key infrastructures where such "poisoned" signatures cannot be attributed to anyone other than the signer herself.) This sort of abuse is possible in all current group signature schemes, unless we make very strong assumptions about underlying communication channels [23]. To this end, immediate-revocation is necessary to block such abuses without having to introduce unrealistic assumptions.

1.1. Our Contributions

This paper makes two noteworthy contributions. First, as mentioned above, we identify two important new security properties: leak-freedom and immediate-revocation. Second, we construct a secure and practical group signature scheme that satisfies both new and existing security properties. The proposed is very efficient: only 11 exponentiations are needed to generate a group signature, while the cost of signature verification is equivalent to verifying a single plain (i.e., non-group) signature, such as RSA. These costs are appreciably lower than those of state-of-the-art group signature schemes [9], [34].

The new scheme also offers two important advantages over previous work:

- It removes the burden of having all signers and all verifiers constantly obtaining and maintaining up-to-date "state information" pertaining to current membership. In prior work, either all signers and all verifiers had to be aware of public revocation lists [7, 2], or all signers had to be aware of public accumulator ingredient lists [9, 34]. In the proposed scheme, the issue of dynamic group membership is fully transparent to both signers (group members) and verifiers.
- It relaxes the requirement for underlying anonymous communication channels, since a group member does not need to communicate directly with signature verifier(s). Whereas, anonymous communication channels are essential (although rarely mentioned explicitly) in previous group signature schemes.

Outline: the rest of this paper is organized as follows: the next section provides the definitions of group signatures and their desired security requirements. Section 3 presents the new group signature scheme. Section 4 discusses several extensions and Section 5 concludes the paper.

2. Definitions

Definition 2.1 *A* **group signature scheme** *includes the following components:*

`Setup:` *a probabilistic polynomial-time algorithm that, on input of a security parameter k, outputs the specification of a cryptographic context including the group manager's public key $pk_{\mathcal{GM}}$ and secret key $sk_{\mathcal{GM}}$.*

`Join:` *a protocol between the group manager $\mathcal{GM}$ and a user that results in the user becoming a group member $\mathcal{U}$. Their common output includes the user's unique membership public key $pk_{\mathcal{U}}$, and perhaps some updated information that indicates the current state of the system. The user's output includes a membership secret key $sk_{\mathcal{U}}$.*

`Revoke:` *an algorithm that, on input of a group member's identity (and perhaps her public key $pk_{\mathcal{U}}$), outputs updated information that indicates the current state of the system after revoking the membership of a given group member.*

`Sign:` *a probabilistic algorithm that, on input of a group public key $pk_{\mathcal{GM}}$, a user's membership secret/public key-pair $(sk_{\mathcal{U}}, pk_{\mathcal{U}})$, and a message m, outputs a group signature δ of m.*

`Verify:` *an algorithm that, on input of a group public key $pk_{\mathcal{GM}}$, a group signature δ, and a message m, outputs a binary value* TRUE/FALSE *indicating whether δ is a valid group signature (under $pk_{\mathcal{GM}}$) of m.*

`Open:` *an algorithm executed by the group manager $\mathcal{GM}$. It takes as input of a message m, a group signature δ, the group public key pk_{GM} and the group manager's secret key $sk_{\mathcal{GM}}$. It first executes* `VERIFY` *on the first three inputs and, if the δ is valid, outputs some incontestable evidence*

1 Except the group manager who can anyway always identify a signer.

(e.g., a membership public key $pk_{\mathcal{U}}$ and a proof) that allows anyone to identify the actual signer.

We now specify the desired security properties of a group signature scheme. These properties are presented informally; a more formal specification appears in the full version of this paper [21]. We note that, although we use the notions of full-traceability and full-anonymity introduced in [4], the definitions of [4] have to be amended to accommodate dynamic groups, since [4] only considers static groups.

Definition 2.2 *A* **secure** *group signature scheme must have the following properties:* correctness*,* full-traceability*,* full-anonymity*,* no-misattribution*, as well as the newly introduced* leak-freedom *and* immediate-revocation*.*

Correctness*: any signature produced by any group member using* `Sign` *must be accepted by* `Verify`*.*

Full-traceability*: no subset of colluding group members (even consisting of the entire group, and even in possession of the group manager's secret key for opening signatures) can create valid signatures that cannot be opened, or signatures that cannot be traced back to some member of the colluding coalition.*

Full-anonymity*: it is computationally infeasible for an adversary (who is not in possession of the group manager's secret key for opening signatures) to recover the identity of the signer from a group signature, even if the adversary has access to the secret keys of all group members.*

No-misattribution*: it is computationally infeasible for a group manager to provably attribute a group signature to a member who is not the actual signer.*

Leak-freedom*: it is computationally infeasible for a signer to convince anyone that it actually signed a given message, even if it possesses all other signers' secrets, except the group manager's secret for opening signatures.*

Immediate-revocation*: it is computationally infeasible for a group member revoked at time t to generate a valid group signature at any time $t' > t$.*

3. Our Construction

Unlike most prior work, our construction is designed from a *systems*, rather than purely cryptographic, perspective. (Nonetheless, cryptography still plays a major role in our construction.) The main idea is the introduction of a new entity referred to as the *mediation server* ($\mathcal{MS}$). $\mathcal{MS}$ is an on-line partially trusted server that helps in signature generation and membership revocation.

Roughly speaking, the system functions as follows: each time a group member needs to generate a signature, it somehow "identifies" itself to the mediation server which then (if the member is not revoked) produces a normal signature. This normal signature is the actual group signature and may be delivered to intended verifier(s). Nevertheless, the mere introduction of the mediation server does not imply that we can trivially obtain a group signature scheme possessing all desired security properties.

Caveat: unlike prior *non-interactive* group signature schemes, our scheme requires some light-weight interaction (which explains why it is able to satisfy all of the requirements). While interaction can be viewed as a drawback, we claim that it constitutes a reasonable (even small) price to pay for additional attained security properties.

3.1. Further Motivation

At this point, one might wonder whether a practical solution that satisfies all aforementioned requirements can be obtained in a trivial fashion. A trivial approach that satisfies all aforementioned requirements is to make the group manager an on-line entity and have it "filter" all group signature requests. Each group member has a private channel to the group manager $\mathcal{GM}$ and, for each message to be signed, it submits a message signed under its normal long-term signature key. $\mathcal{GM}$ then "translates" each incoming signature into a signature under its own well-known group signature key. The latter is then released to the verifier(s) and treated as a group signature. With this trivial solution all security properties (including leak-freedom and immediate-revocation) are satisfied and signature generation/verification costs are minimal.

However, the trivial approach requires constant ironclad security of the $\mathcal{GM}$. If this can be assured, then the trivial solution is perfect. However, having a **fully trusted on-line** entity is undesirable and sometimes simply unrealistic.[2] A on-line $\mathcal{GM}$ would be a single point of failure in terms of both security (i.e., compromise of $\mathcal{GM}$ means compromise of the whole system) and anonymity (i.e., a dishonest $\mathcal{GM}$ can arbitrarily "open" a group signature without being held accountable). It essentially "puts all eggs in one basket." One standard way to avoid a single point of failure is to utilize distributed cryptography, which usually takes a heavy toll in system complexity, including: management, computation and communication. To avoid unnecessary complexity and subtleties, we design a system under the guidance of the well-known separation of duty principle. (See the seminal work of Clark and Wilson [20] for necessary background.) This approach, as will be shown below, facilitates a practical solution with a similar flavor of distributed security, i.e., compromise of either $\mathcal{GM}$ or the newly introduced

2 For much the same reasons that Certification Authorities (CAs) are not on-line.

mediation server $\mathcal{MS}$, but not both, does not imply complete compromise of the system.

3.2. Model

PARTICIPANTS: a set of group members $\mathbb{U}$ where $|\mathbb{U}| = n$, a group manager $\mathcal{GM}$ who admits group members, a mediation server $\mathcal{MS}$ who revokes group members (according to $\mathcal{GM}$'s instructions) and a set of signature receivers. Each participant is modeled as a probabilistic polynomial-time interactive Turing machine. We assume that $\mathcal{MS}$ maintains a dynamic membership database DBMember-MS and $\mathcal{GM}$ maintains a similar dynamic membership database DBUser-GM. Moreover, we assume that $\mathcal{MS}$ maintains a database DBSig-MS that records all signature transactions. We further assume that this database is never compromised. In practice, various uncomplicated protection mechanisms can be utilized.[3]

COMMUNICATION CHANNELS: all communication channels are asynchronous and under adversary's control, except the channel between $\mathcal{GM}$ and $\mathcal{MS}$. In a typical system configuration, we do not assume any anonymous channels. However, we do assume that the channels between a group member $\mathcal{U} \in \mathbb{U}$ and $\mathcal{GM}$, and between $\mathcal{GM}$ and $\mathcal{MS}$ are authentic. (This can be implemented via standard techniques and is thus ignored in the rest of the paper).

TRUST: precise specification of the trust model is admittedly difficult mainly because of the introduction of the new party ($\mathcal{MS}$). Nevertheless, taking into account the *separation-of-duty* principle, we have the following:

1. $\mathcal{GM}$ is trusted not to introduce any illegal (or phantom) group members. However, $\mathcal{GM}$ may want to frame an honest group member.
2. $\mathcal{MS}$ is trusted to enforce $\mathcal{GM}$'s policy, e.g., to refuse services to any and all revoked group members; equivalently, to produce group signatures only for legitimate members. $\mathcal{MS}$ is assumed not to misbehave if such activity will be traced to it. Nonetheless, $\mathcal{MS}$ may attempt to: 1) frame an honest member into signing a message, 2) generate an unattributable group signature, and 3) compromise anonymity of an honest group member (e.g., via out-of-band means).

SECURITY DEFINITIONS: in order to accommodate the new entity ($\mathcal{MS}$), we need to slightly amend some parts of Definition 2.2 for our setting. The changes are minimal but necessary for the sake of clarity. (The rest of the definitions remains unchanged.)

Definition 3.1 Full-traceability*: the set of colluders can additionally include $\mathcal{MS}$.*
Full-anonymity*: the adversary is additionally allowed to have access to the secret key(s) of $\mathcal{MS}$.*
Leak-freedom*: it is infeasible for a signer to convince anyone (except $\mathcal{MS}$) that it generated a group signature; it is infeasible for $\mathcal{MS}$ to convince anyone (except $\mathcal{GM}$) that a certain group member generated a group signature.*

3.3. Accountable Designated Verifier Signatures

We introduce the notion of accountable designated verifier signatures (ADVS) that will serve as the building block in our group signature scheme. This notion is an enhancement of the private contract signatures introduced in [25]. Informally, a private contract signature is a designated verifier signature that can be converted into a universally-verifiable signature by either the signer or a trusted third party appointed by the signer. An **accountable** designated verifier signature scheme, on the other hand, emphasizes the trusted third party's capability to identify the actual signer of a valid signature.

Definition 3.2 *Let P_i and P_j be two distinct entities and T be a trusted third party. Suppose P_i is the signer and P_j is the verifier. An accountable designated verifier signature scheme,* ADVS*, is a tuple of polynomial-time algorithms (*ADVS-Sign*,* ADVS-Ver*,* ADVS-Proof*) defined as follows:*

ADVS-Sign*: an algorithm executed by P_i on message m to output an ADVS: $\delta = \texttt{ADVS-Sign}_{P_i}(m, P_j, T)$.*

ADVS-Ver*: an algorithm which allows P_j to verify the validity of an input tag δ on message m, such that:*
$\texttt{ADVS-Ver}(m, P_i, P_j, T; \delta) =$
$$\begin{cases} \text{TRUE} & if\ \delta = \texttt{ADVS-Sign}_{P_i}(m, P_j, T) \\ \text{FALSE} & otherwise \end{cases}$$

ADVS-Proof*: an algorithm executed by T on input P_i, m, P_j, and tag δ. It outputs a proof for the predicate* SignedBy(δ, P_i) *which is* TRUE *if δ is produced by P_i, and* FALSE *otherwise.*

We require that, if $\delta = \texttt{ADVS-Sign}_{P_i}(m, P_j, T)$, then: $\texttt{ADVS-Ver}(m, P_i, P_j, T; \delta) =$ TRUE *and T always outputs a proof for* SignedBy(δ, P_i) *being* TRUE*.*

Definition 3.3 *An ADVS scheme is* **secure** *if the following properties are satisfied:*

Unforgeability *of* $\texttt{ADVS-Sign}_{P_i}(m, P_j, T)$*: for any m, it is infeasible for any $P \notin \{P_i, P_j\}$ to produce δ, such that* $\texttt{ADVS-Ver}(m, P_i, P_j, T; \delta) =$ TRUE*.*

3 For example, $\mathcal{MS}$ can use a *decoy* technique and insert $(n-1)$ well-formed dummy transaction records for each genuine one. Alternatively, the database can be kept encrypted with the encryption key protected using some advanced cryptographic techniques. These and other approaches and their respective merits are discussed in the full version of this paper [21].

Non-transferability *of* $\texttt{ADVS-Sign}_{P_i}(m, P_j, T)$*: for* P_j*, there is a polynomial-time forgery algorithm which, for any* m*,* P_i*, and* T*, outputs* δ *such that* $\texttt{ADVS-Ver}(m, P_i, P_j, T; \delta) =$ TRUE.

Unforgeability *of the proof for* SignedBy(δ, P_i)*: for any* $\delta = \texttt{ADVS-Sign}_{P_i}(m, P_j, T)$*, it is infeasible for any* $P \notin \{T, P_i\}$ *to produce a proof for* SignedBy(δ, P_i) *being* TRUE.

We need to further differentiate between the case that P_i is able to produce a proof for SignedBy(δ, P_i) and the case that P_i is unable to do so. Although the former is desirable in the context of private contract signatures, it is undesirable in our setting. This is because private contract signatures only intend to prevent P_j transferring the one bit information SignedBy(δ, P_i) by any means. Whereas, it would be very useful in our context for both P_i and P_j not to be able to leak the bit: SignedBy(δ, P_i). Thus, we have the following definition:

Definition 3.4 *An ADVS scheme is* strongly-secure *if, in addition to being a* secure *ADVS scheme, it ensures that a signer* P_i *cannot produce a proof for* SignedBy(δ, P_i) *with non-negligible probability, where* $\delta = \texttt{ADVS-Sign}_{P_i}(m, P_j, T)$.

A *strongly-secure* ADVS scheme can allow us to easily construct a simple leak-free group signature system. Unfortunately, we do not know how to construct such a scheme, so we leave it as an interesting open problem. In order to facilitate a group signature scheme that is leak-free with immediate-revocation, we utilize a *secure* ADVS scheme that is based on the ideas in [25]. Due to space limitation, the details appear in the full version of this paper [21].

3.4. Leak-free Group Signatures with Immediate Revocation

We are finally ready to present a concrete construction. The basic operation of our scheme is as follows. Given a message m, a group member $\mathcal{U}_i$ presents an ADVS $\delta = \texttt{ADVS-Sign}_{\mathcal{U}_i}(m, \mathcal{MS}, \mathcal{GM})$ to $\mathcal{MS}$ requesting (and obtaining) a plain signature $\sigma = \texttt{Sign}_{\mathcal{MS}}(m)$. The latter is viewed as a group signature, for which there is a single group-wide verification key. Note that, since $\mathcal{GM}$ plays the role of a trusted third party in the ADVS scheme, it can hold an actual signer accountable. We also note that our trust model implies that there are no issues with the fair exchange of $\delta = \texttt{ADVS-Sign}_{\mathcal{U}_i}(m, \mathcal{MS}, \mathcal{GM})$ for $\sigma = \texttt{Sign}_{\mathcal{MS}}(m)$, even if σ is returned to the user. Following the definition of group signatures, our mediated group signature scheme consists of the following procedures:

SETUP: this procedure initializes a group manager $\mathcal{GM}$ and a mediation server $\mathcal{MS}$.

1. $\mathcal{GM}$chooses a system wide security parameter κ. Based on it, $\mathcal{GM}$chooses a discrete-logarithm based crypto-context as in a normal Schnorr signature setting. The parameter κ and the crypto-context are followed system-wide by all group members, the $\mathcal{MS}$ and $\mathcal{GM}$ itself. $\mathcal{GM}$initializes a set $\mathbb{U}$ wherein each group member will be assigned a unique identity $\mathcal{U}_i \in \mathbb{U}$. $\mathcal{GM}$then instantiates an ADVS scheme ADVS as well as its own public/private key-pair: $\langle Y_{\mathcal{GM}} = g^{X_{\mathcal{GM}}}, X_{\mathcal{GM}} \rangle$. Finally, $\mathcal{GM}$initializes the database DBUser-GM.
2. The initialization of the mediation server $\mathcal{MS}$ consists of the following: choose a pair of ADVS public and private keys $\langle Y_{\mathcal{MS}} = g^{X_{\mathcal{MS}}}, X_{\mathcal{MS}} \rangle$ in the same crypto-context as in step 1. Choose a pair of keys for a normal digital signature scheme $\mathcal{SIG} = (\texttt{Gen}, \texttt{Sign}_{\mathcal{MS}}, \texttt{Ver}_{\mathcal{MS}})$ that is secure against adaptive chosen-message attack [27]. Denote by $\langle pk_{\mathcal{MS}}, sk_{\mathcal{MS}} \rangle$ the pair of group signature verification and generation keys, where $pk_{\mathcal{MS}}$ is publicly known.[4] Initialize databases DBMember-MS and DBSig-MS.

JOIN: whenever $\mathcal{GM}$ decides to admit a new member, it first assigns a unique identity $\mathcal{U}_i$. $\mathcal{U}_i$ generates its ADVS public/private key-pair: $\langle Y_{\mathcal{U}_i} = g^{X_{\mathcal{U}_i}}, X_{\mathcal{U}_i} \rangle$. $Y_{\mathcal{U}_i}$ is then registered at the $\mathcal{GM}$ and $\mathcal{MS}$; both DBMember-GM and DBMember-MS are updated accordingly.

SIGN: whenever a group member $\mathcal{U}_i$ wants to generate a group signature on message m, the following protocol is executed:

1. $\mathcal{U}_i$ sends $\mathcal{MS}$ an ADVS $\delta = \texttt{ADVS-Sign}_{\mathcal{U}_i}(m, \mathcal{MS}, \mathcal{GM})$ over an insecure public (and unauthenticated) channel.
2. Upon receipt, $\mathcal{MS}$ retrieves $\mathcal{U}_i$'s public key $Y_{\mathcal{U}_i}$ from its database DBMember-MS. If no entry is found, $\mathcal{MS}$ simply ignores the request; otherwise, $\mathcal{MS}$ verifies δ by checking whether $\texttt{ADVS-Ver}(m, \mathcal{MS}, \mathcal{GM}; \delta) =$ TRUE. If it is, $\mathcal{MS}$ then produces a normal signature $\gamma = \texttt{Sign}_{\mathcal{MS}}(m)$ and inserts a new record $(\mathcal{U}_i, \delta, \gamma)$ into its database DBSig-MS. The signature γ is a group signature on message m.

VERIFY: given $pk_{\mathcal{MS}}$, the public group signature verification key of $\mathcal{MS}$, and a tag γ, anyone can establish whether γ is a valid (group) signature by running $\texttt{Ver}_{\mathcal{MS}}$ on input: $pk_{\mathcal{MS}}$, m, and γ.

4 We assume that $\mathcal{MS}$ knows $sk_{\mathcal{MS}}$ in its entirety so as to prevent attacks due to incorrect system initialization. This can be ensured by utilizing techniques due to [35].

REVOKE: whenever $\mathcal{GM}$ decides to revoke membership of $\mathcal{U}_i$, it first updates DBUser-GM indicating that $\mathcal{U}_i$ is revoked and logs the necessary information. $\mathcal{GM}$ then informs $\mathcal{MS}$ that $\mathcal{U}_i$ is revoked. and $\mathcal{MS}$ proceeds to delete the entry $(\mathcal{U}_i, Y_{\mathcal{U}_i})$ from its database DBMember-MS. Consequently, all subsequent signature requests from $\mathcal{U}_i$ will be rejected by $\mathcal{MS}$.

OPEN: whenever $\mathcal{GM}$ decides to identify the actual signer of signature γ on message m, the following protocol is executed by $\mathcal{GM}$ and $\mathcal{MS}$:

1. $\mathcal{GM}$ sends γ to $\mathcal{MS}$ via an authenticated channel.
2. $\mathcal{MS}$ retrieves $(\mathcal{U}_i, \delta, \gamma)$ from its database and sends it to $\mathcal{GM}$ via the same authenticated channel.
3. $\mathcal{GM}$ checks whether $\texttt{ADVS-Ver}(m, \mathcal{MS}, \mathcal{GM}; \delta) =$ TRUE. If so, $\mathcal{GM}$ executes ADVS-Proof to produce a proof for $\text{SignedBy}(\delta, \mathcal{U}_i)$; otherwise, $\mathcal{GM}$ concludes that $\mathcal{MS}$ is cheating.

Remark: the above protocol does not specify how a group signature is sent to its intended verifier(s). There are two options. One way that allows us to completely get rid of the anonymous channels is to let $\mathcal{MS}$ send γ directly to the verifier(s). In this case, there may be a need for a random delay to address potential traffic analysis (see further discussion below). Note that this kind of delay already exists in the anonymous channels currently available. The other way is to let $\mathcal{MS}$ broadcast γ so that $\mathcal{U}_i$ can receive and re-send γ to the verifier(s) via an anonymous channel. We prefer the former, however, a detailed analysis is deferred to [21].

3.5. Security Analysis

Theorem 3.1 *Our scheme satisfies the requirements specified in Definition 3.1, namely* correctness*,* full-traceability*,* full-anonymity*,* no-misattribution*,* leak-freedom*, and* immediate-revocation*.*

A formal proof of this theorem appears in a full version of this paper [21]. Intuitively, the theorem holds because the final group signatures are generated using a standard signature algorithm, and ADVS makes it infeasible for the group members or $\mathcal{MS}$ to cryptographically prove the linkage between a signature request and the resulting group signature.

4. Extension and Discussion

Enhancing anonymity against traffic analysis. Our scheme does not assume that the channel between a group member $\mathcal{U}$ and the mediation server $\mathcal{MS}$ is authenticated or anonymous. This is because $\mathcal{MS}$ may have incentives to cheat an outsider, which, in turn, implies:

1. Even if the adversary eavesdrops on all channels, there could still be an out-of-band channel between a group member and $\mathcal{MS}$. Thus, the adversary could still be fooled.
2. $\mathcal{MS}$ can easily cheat an outsider by injecting a fake ADVS into the network or its database DBSig-MS.

However, an adversary might know that $\mathcal{MS}$, while not being trusted to preserve anonymity, does not always inject fake traffic into the network. Then, an adversary has a good chance of compromising anonymity of some honest group members by means of traffic analysis. Fortunately, this can be avoided by using standard techniques, such as end-to-end encryption and traffic padding.

On strongly-secure ADVS vs. secure ADVS. In our construction we utilized an ADVS that is *secure*, but not *strongly-secure*. Consequently, we assume the secrecy of $\mathcal{MS}$'s storage, particularly DBSig-MS consisting of all signature transaction entries. This is necessary in order to avoid the following attack: If an attacker has access to an entry in DBSig-MS, then Alice can convince the attacker that she generated a group signature $\texttt{Sign}_{\mathcal{MS}}(m)$. Clearly, if a *strongly-secure* ADVS is utilized, we can achieve strictly stronger security that Alice is unable to convince XYZ that she generated a signature, even if XYZ has access to the corresponding entry in DBSig-MS. It seems non-trivial to ensure secrecy of DBSig-MS, while preserving other desirable properties. We continue investigating technical mechanisms that can replace this rather strong assumption regarding database secrecy [21].

Denial-of-Service (DoS) attacks. Recall than $\mathcal{MS}$ always performs a number of modular exponentiations before it is able to determine whether an incoming ADVS is valid. This opens the door for DoS attacks on $\mathcal{MS}$. To counter such attacks, we suggest a simple solution. The idea is to let each $\mathcal{U}_i$ and $\mathcal{MS}$ share a unique secret key w_i. Each signature request from $\mathcal{U}_i$ must also be accompanied by an authentication token computed over the request with the key w_i. When processing a request, $\mathcal{MS}$ verifies the authentication token before performing much more expensive validation of the ADVS. Note that this additional authentication is does not in any way influence security properties of our scheme. Of course, this is only a partial solution since an adversary can still mount a DoS attack on the $\mathcal{MS}$'s network interface.

4.1. Related Work

This paper can be viewed as one of many efforts pursuing practical and secure *group signature* or *identity escrow* schemes [18, 19, 13, 29, 1, 11], as well as anonymous *credential systems* [16, 17, 30, 10, 14]. Among them, the prior work most relevant to this paper is [11], which presented an

identity escrow scheme (and a corresponding group signature scheme) with the appointed verifier property. The motivation was to obtain a scheme where a group member can only convince one or more appointed verifiers of its membership, while no other party can verify membership even if the signer cooperates fully.

Note that there is a difference between the appointed verifier property in [11] and the leak-freedom property specified in this paper. Specifically, the scheme in [11], by definition, allows a signer to convince the designated verifiers that it is authorized to conduct the relevant transactions. Cast this in our example scenario, Alice can always convince XYZ that she is authorized to sign purchase orders. This capability can result in the leakage (outlined in Section 1) we meant to avoid! Besides achieving the strictly stronger leak-freedom, our scheme is more efficient than [11] which requires both a signer and a verifier to compute more than $17k$ exponentiations, where k is a security parameter (say, $k = 80$). Moreover, membership revocation is not supported in [11], whereas, we achieve immediate-revocation which has only been explored in the context of traditional PKI-s [6].

A credential system is a system where users can obtain credentials from organizations and demonstrate possession of these credentials [16, 17, 30, 10, 14]. Chaum and Evertse [17] presented a general scheme using a semi-trusted TTP common to multiple organizations. However, their scheme is impractical. The credential system by Lysianskaya et al. [30] captures many of the desirable properties. Camenisch and Lysianskaya [10] presented a better solution with ingredients from a secure group signature scheme of [1]. The prototype implementation of [10] was done by Camenisch and van Herreweghen [14]. This scheme requires both signers and verifiers to compute 22 modular exponentiations. Their advanced scheme which provides all-or-nothing non-transferability (to discourage a signer from sharing its credentials with other parties) requires both signer and verifier to compute 200 exponentiations.

The notion of *abuse-freedom* or abuse-freeness [25] is weaker than leak-freedom because the former intends only to prevent the *designated verifier* from being able to transfer the information about the actual signer, whereas the latter intends to prevent a *signer* as well as the *designated verifier* from being able to transfer the same information. Finally, we remark that leak-freedom is similar to the property called *receipt-freedom* or receipt-freeness in the context of voting schemes [5, 28], and its closely related variants called deniable encryption [15] and deniable proof [24].

5. Conclusion

We identified two properties, namely leak-freedom and immediate-revocation, that are crucial for a large class of group signature applications. We also constructed a scheme that achieves all of traditional and newly-introduced goals by following a *systems architectural approach*. Our scheme is practical and easy to implement, because it needs only 11 exponentiations for a group member to generate a group signature and one normal signature verification for its validation. Another contribution of this paper is the relaxation on the requirement for anonymous communication channels, which are essential in all previous schemes.

In addition to the already mentioned issues (including the protection of the server databases, and the robustness against denial-of-service attacks) that need further investigations, our work also incurs some problems that are interesting in an even more general context:

1. How to construct a practical *strongly-secure* ADVS scheme?
2. How to construct a leak-free group signature scheme with immediate revocation that does not rely on a mediation server? Although we believe that the existence of a mediation server is more realistic than the existence of (for instance) a time-stamping service, it is nevertheless conceivable that other alternatively constructions could fit well into different specific application scenarios.
3. How to achieve a stateless $\mathcal{MS}$? This is not trivial because, otherwise, the binding of an ADVS signature (or any other token with the desired properties) and a normal signature would allow $\mathcal{MS}$ to convince an outsider of the identity of the actual signer.

References

[1] G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A practical and provably secure coalition-resistant group signature scheme. In *CRYPTO'2000*, pages 255–270.

[2] G. Ateniese, D. Song, and G. Tsudik. Quasi-efficient revocation of group signatures. In *Financial Crypto'02)*.

[3] G. Ateniese and G. Tsudik. Some open issues and new directions in group signatures. In *Financial Crypto'99)*.

[4] M. Bellare, D. Micciancio, and B. Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In *EuroCrypto'2003*.

[5] J. Benaloh and D. Tuinstra. Receipt-free secret-ballot election (extended abstract). In *STOC*, 1994.

[6] D. Boneh, X. Ding, G. Tsudik, and C. Wong. A method for fast revocation of public key certificates and security capabilities. In *10th USENIX Security Symposium*, 2001.

[7] E. Bresson and J. Stern. Group signatures with efficient revocation. In *International Workshop on Practice and Theory in Public Key Cryptography (PKC) '2001*.

[8] J. Camenisch. *Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem*. PhD thesis, ETH Zurich, 1998.

Fine-grained Control of Security Capabilities

Dan Boneh
Computer Science Department, Stanford University. dabo@cs.stanford.edu
and
Xuhua Ding
Department of Information and Computer Science, University of California, Irvine. xhding@ics.uci.edu
and
Gene Tsudik
Department of Information and Computer Science, University of California, Irvine. gts@ics.uci.edu

We present a new approach for fine-grained control over users' security privileges (fast revocation of credentials) centered around the concept of an on-line semi-trusted mediator (SEM). The use of a SEM in conjunction with a simple threshold variant of the RSA cryptosystem (mediated RSA) offers a number of practical advantages over current revocation techniques. The benefits include simplified validation of digital signatures, efficient certificate revocation for legacy systems and fast revocation of signature and decryption capabilities. This paper discusses both the architecture and the implementation of our approach as well as its performance and compatibility with the existing infrastructure. Experimental results demonstrate its practical aspects.

Categories and Subject Descriptors: E.3.3 [**Data**]: Data Encryption—*Public Key Cryptosystems*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms: Algorithms, Security
Additional Key Words and Phrases: Certificate Revocation, Digital Signatures, Public Key Infrastructure

1. INTRODUCTION

We begin this paper with an example to illustrate the premise for this work. Consider an organization – industrial, government, or military – where all employees (referred to as *users*) have certain authorizations. We assume that a Public Key Infrastructure (PKI) is available and all users have digital signature, as well as en/de-cryption, capabilities. In the course of performing routine everyday tasks, users take advantage of secure applications, such as email, file transfer, remote log-in and web browsing.

Now suppose that a trusted user (Alice) does something that warrants immediate revocation of her security privileges. For example, Alice might be fired, or she may suspect that her private key has been compromised. Ideally, immediately following revocation, the key holder, either Alice herself or an attacker, should be unable to perform any security operations and use any secure applications. Specifically, this might mean:

– The key holder cannot read any secure email. This includes encrypted email that already resides on Alice's email server (or local host) and possible future email erroneously encrypted for Alice. Although encrypted email may be delivered to Alice's email server, the key holder should be unable to decrypt it.
– The key holder cannot generate valid digital signatures on any further messages. However, signatures generated by Alice prior to revocation may need to remain valid.
– The key holder cannot authenticate itself to corporate servers (and other users) as a legitimate user.

Throughout the paper, we use email as an example application. While it is a popular mechanism for general-purpose communication, our rationale also applies to other secure means of information exchange.

To provide immediate revocation it is natural to first consider traditional revocation techniques. Many revocation methods have been proposed; they can be roughly classified into two prominent types: 1) explicit revocation structures such as *Certificate Revocation Lists (CRLs)* and variations on the theme, and 2) real time revocation checking such as the *Online Certificate Status Protocol (OCSP)* [Myers et al. 1999] and its variants. In both cases, some trusted entities are ultimately in charge of validating user certificates. However, the above requirements for immediate revocation are impossible to satisfy with existing techniques. This is primarily because they do not provide fine-grained enough control over users' security capabilities. Supporting immediate revocation with existing revocation techniques would result in heavy performance cost and very poor scalability, as discussed in Section 8.

As pointed out in [McDaniel and Rubin 2000], since each revocation technique exhibits a unique set of pros and cons, the criteria for choosing the best technique should be based on the specifics of the target application environment. Fast revocation and fine-grained control over users' security capabilities are the motivating factors for our work. However, the need for these features is clearly not universal since many computing environments (e.g., a typical university campus) are relatively "relaxed" and do not warrant employing fast revocation techniques. However, there are plenty of government, corporate and military settings where fast revocation and fine-grained control are very important.

Organization. This paper is organized as follows. The next section provides an overview of our work. The technical details of the architecture are presented in Section 3 and Section 4, respectively. Then, Section 5 shows four extensions. Sections 6 and 7, describe the implementation and performance results, respectively. A comparison of with current revocation techniques is presented Section 8, followed by the overview of related work in Section 8.2 and a summary in Section 9.

2. OVERVIEW

We refer to our approach as the SEM **architecture**. The basic idea is as follows: We introduce a new entity, referred to as a SEM (SEcurity Mediator): an online semi-trusted server. To sign or decrypt a message, a client must first obtain a message-specific token from its SEM. Without this token, the user cannot accomplish the intended task. To revoke the user's ability to sign or decrypt, the security administrator instructs the SEM to stop issuing tokens for that user's future request. At that instant, the user's signature and/or decryption capabilities are revoked. For scalability reasons, a single SEM serves many users.

We stress that the SEM architecture is transparent to non-SEM users, i.e., a SEM is not involved in encryption or signature verification operations. With SEM's help, a SEM client (Alice) can generate standard RSA signatures, and decrypt standard ciphertext messages encrypted with her RSA public key. Without SEM's help, she cannot perform either of these operations. This backwards compatibility is one of our main design principles.

Another notable feature is that a SEM is not a *fully* trusted entity. It keeps no client secrets and all SEM computations are checkable by its clients. However, a SEM is *partially trusted* since each signature verifier implicitly trusts it to have checked the signer's (SEM's client's) certificate status at signature generation time. Similarly, each encryptor trusts a SEM to check the decryptor's (SEM's client's) certificate status at message decryption time. We consider this level of trust reasonable, especially since a SEM serves a multitude of clients and thus represents an organization (or a group).

In order to experiment and gain practical experience, we prototyped the SEM architecture using the popular OpenSSL library. SEM is implemented as a daemon process running on a secure server. On the client side, we built plug-ins for the Eudora and Outlook email clients for signing outgoing, and decrypting incoming, emails. Both of these tasks are performed with the SEM's help. Consequently, signing and decryption capabilities can be easily revoked.

It is natural to ask whether the same functionality can be obtained with more traditional security approaches to fine-grained control and fast credential revocation, such as Kerberos. Kerberos [Neuman and Ts'o 1994], after all, has been in existence since the mid-80s and tends to work very well in corporate-style settings. However, Kerberos is awkward in heterogeneous networks such as the Internet; its inter-realm extensions are difficult to use and require a certain amount of manual setup. Furthermore, Kerberos does not inter-operate with modern PKI-s and does not provide universal origin authentication offered by public key signatures. On the other hand, the SEM architecture is fully compatible with existing PKI systems. In addition, the SEM is only responsible for revocation. Unlike a Kerberos server, the SEM cannot forge user signatures or decrypt messages intended for users. As we discuss later in the paper, our approach is not mutually exclusive with Kerberos-like intra-domain security architectures. We assert that the SEM architecture can be viewed as a set of complementary security services.

2.1 Decryption and signing in the SEM architecture

We now describe in more detail how decryption and digital signature generation are performed in the SEM architecture:
– Decryption: suppose that Alice wants to decrypt an email message using her private key. Recall that public key-encrypted email is usually composed of two parts: (1) a short preamble containing a per-message key encrypted with Alice's public key, and (2) the body containing the actual email message encrypted using the per-message key. To decrypt, Alice first sends the preamble to her SEM. SEM responds with a token which enables Alice to complete the decryption of the per-message key and, ultimately, to read her email. However, this token contains no information useful to anyone other than Alice. Hence, communication with the SEM does not need to be secret or authenticated. Also, interaction with the SEM is fully managed by Alice's email reader and does not require any intervention on Alice's part. If Alice wants to read her email offline, the interaction with the SEM takes places at the time Alice's email client downloads her email from the mail server.

– Signatures: suppose that Alice wishes to sign a message using her private key. She sends a (randomized) hash of the message to her SEM which, in turn, responds with a token (also referred to as a half-signature) enabling Alice to generate the signature. As with decryption, this token contains no useful information to anyone other than Alice.

2.2 Other Features

Our initial motivation for introducing a SEM is to enable immediate revocation of Alice's public key. As a byproduct, the SEM architecture provides additional benefits. In particular, validation of signatures generated with the help of a SEM does not require the verifier to consult a CRL or a revocation authority: the existence of the a verifiable signature implies that the signer was not revoked at the time the signature was generated. Consequently, signature validation is greatly simplified.

More importantly, the SEM architecture enables revocation in legacy systems that do not support certificate revocation. Consider a legacy system performing digital signature verification. Often, such systems have no certificate status checking capabilities. For example, old browsers (e.g., Netscape 3.0) verify server certificates without any means for checking certificate revocation status. Similarly, Microsoft's Authenticode system in Windows NT (used for verifying signatures on executable code) does not support certificate revocation. In the SEM architecture, certificate revocation is provided without requiring any change to the verification process in such legacy systems. The only aspect that needs changing is signature generation. Fortunately, in many settings (such as code signing) the number of entities generating signatures is significantly smaller than that of entities verifying them.

Semantics. The SEM architecture naturally provides the following semantics for digital signatures:

> **Binding Signature Semantics:** *a digital signature is considered valid if the public key certificate associated with the private key used to generate the signature was valid at the time the signature was issued.*

According to this definition, all verifiable signatures – by virtue of their existence – are generated prior to revocation and, hence, are considered valid. Binding signature semantics are natural in many settings, such as business contracts. For example, suppose Alice and Bob enter into a contract. They both sign the contract at time T. Bob begins to fulfill the contract and incurs certain costs in the process. Now, suppose at time $T' > T$, Alice revokes her own certificate (e.g., by "losing" her private key). Is the contract valid at time T'? With binding semantics, Alice is still bound to the contract since it was signed at time T when her certificate was still valid. In other words, Alice cannot nullify the contract by causing her own certificate to be revoked. We note that binding semantics are inappropriate in some scenarios. For example, if a certificate is obtained from a CA under false pretense, e.g., Alice masquerading as Bob, the CA should be allowed to declare at any time that **all** signatures generated with that certificate are invalid.

Implementing binding signature semantics with existing revocation techniques is non-trivial, as discussed in Section 8. Whenever Bob verifies a signature generated by Alice, Bob must also check that Alice's certificate was valid at the time the signature was generated. In fact, every verifier of Alice's signature must perform this certificate validation step. Note that, unless a trusted time-stamping service is involved in generating all of Alice's signatures, Bob cannot trust the timestamp included by Alice in her signatures.

Not surprisingly, implementing binding semantics with the SEM architecture is trivial. To validate Alice's signature, a verifier need only verify the signature itself. There is no need to check the status of Alice's certificate. (We are assuming here that revocation of Alice's key is equivalent to revocation of Alice's certificate. In general, however, Alice's certificate may encode many rights, not just the right to use her key(s). It is then possible to revoke only some of these rights while not revoking the entire certificate.) Once Alice's certificate is revoked, she can no longer generate valid signatures. Therefore, the mere existence of a valid signature implies that Alice's certificate was valid at the time the signature was issued.

3. MEDIATED RSA

We now describe in detail how a SEM interacts with clients to generate tokens. The SEM architecture is based on a variant of RSA which we call Mediated RSA (mRSA). The main idea is to split each RSA private key into two parts using simple 2-out-of-2 threshold RSA [Gemmel 1997; Boyd 1989]. One part is given to a client and the other is given to a SEM. If the client and its SEM cooperate, they employ their respective half-keys in a way that is functionally equivalent to (and indistinguishable from) standard RSA. The fact that the private key is not held in its entirety by any one party is transparent to the outside world, i.e., to the those who use the corresponding public key. Also, knowledge of a half-key cannot be used to derive the entire private key. Therefore, neither the client nor the SEM can decrypt or sign a message without mutual consent. (Recall that a single SEM serves many clients.)

The mRSA method is composed of three algorithms: mRSA key generation, mRSA signatures, and mRSA decryption. We present them in the next section.

3.1 mRSA Key Generation

Similar to RSA, each client U_i has a unique public key and private key. The public key PK_i includes n_i and e_i, where the former is a product of two large distinct primes (p_i,q_i) and e_i is an integer relatively prime to $\phi(n_i) = (p_i - 1)(q_i - 1)$.

Logically, there is also a corresponding RSA private key $SK_i = (n_i, d_i)$ where $d_i * e_i = 1 \bmod \phi(n_i)$. However, as mentioned above, no one party has possession of d_i. Instead, d_i is effectively split into two parts: d_i^u and d_i^{sem} which are secretly held by the client U_i and a SEM, respectively. The relationship among them is:

$$d_i \;=\; d_i^{sem} \;+\; d_i^u \;\bmod\; \phi(n_i)$$

Unlike plain RSA, an individual client U_i cannot generate its own mRSA keys. Instead, a trusted party (most likely, a CA) initializes and distributes the mRSA keys to clients. The policy for authenticating and authorizing clients' key generation requests is not discussed in this paper. Once a client's request is received and approved, a CA executes the mRSA key generation algorithm described below.

mRSA Key Setup. CA generates a distinct set: $\{p_i,\ q_i,\ e_i,\ d_i,\ d_i^{sem},\ d_i^u\}$ for U_i. The first four values are generated as in standard RSA. The fifth value, d_i^{sem}, is a random integer in the interval $[1, n_i]$, where $n_i = p_i q_i$. The last value is set as: $d_i^u = d_i - d_i^{sem} \bmod \phi(n_i)$. We show the protocol in Figure 1.

Algorithm: mRSA.key (executed by CA)

Let k (even) be a security parameter

(1) Generate random $k/2$-bit primes: p_i, q_i
(2) $n_i \leftarrow p_i q_i$
(3) $e_i \stackrel{r}{\leftarrow} Z^*_{\phi(n_i)}$
(4) $d_i \leftarrow 1/e_i \bmod \phi(n_i)$
(5) $d_i^{sem} \stackrel{r}{\leftarrow} \{1, \ldots, n_i - 1\}$
(6) $d_i^u \leftarrow (d_i - d_i^{sem}) \bmod \phi(n_i)$
(7) $SK_i \leftarrow (n_i, d_i^u)$
(8) $PK_i \leftarrow (n_i, e_i)$

Fig. 1. mRSA Key Generation Algorithm

After CA computes the above values, d_i^{sem} is securely communicated to the SEM and d_i^u is communicated to U_i. The details of this step are elaborated upon in Section 6.

3.2 mRSA Signatures

According to PKCS1 v2.1 [Labs 2002], RSA signature generation is composed of two steps: message encoding and cryptographic primitive computation. The first step is preserved in mRSA without any changes. However, the second step requires SEM's involvement since, in mRSA, a client does not possess its entire private key.

We denote by $EC()$ and $DC()$ the encoding and decoding functions, respectively. Both encodings include hashing the input message m using a collision resistant hash function. For now, we assume the message encoding function $EC()$ is **deterministic**. A user (U_i) generates a signature on a message m as follows:

1. Preprocessing: U_i sends the message m to the SEM.
2. Computation:
   – SEM checks that U_i is not revoked and, if so, computes a partial signature $PS_{sem} = EC(m)^{d_i^{sem}} \bmod n_i$ and replies with it to the client. This PS_{sem} is the token enabling signature generation on message m.
   – Concurrently, U_i computes $PS_u = EC(m)^{d_i^u} \bmod n_i$
3. Output: U_i receives PS_{sem} and computes $S_i(m) = (PS_{sem} * PS_u) \bmod n_i$. It then verifies $S_i(m)$ as a standard RSA signature. (This step also verifies the SEM's computation.) If the signature is valid, U_i outputs it.

The algorithm is shown in Figure 2.

Algorithm mRSA.sign (executed by User and SEM)

(1) USER: Send m to SEM.
(2) In parallel:
  2.1. SEM:
  (a) If USER revoked return (ERROR);
  (b) $PS_{sem} \leftarrow EC(m)^{d_i^{sem}} \bmod n_i$
  where EC() is the EMSA-PKCS1-v1_5 encoding function, recommended in [Labs 2002].
  (c) send PS_{sem} to USER
  2.2. USER:
  (a) $PS_u \leftarrow EC(m)^{d_i^u} \bmod n_i$
(3) USER: $S \leftarrow PS_{sem} * PS_u \bmod n_i$
(4) USER: Verify that S is a valid signature on m under the public key (N, e_i). If not then return (ERROR)
(5) USER: return (m,S)

Fig. 2. mRSA Signature Algorithm

We observe that the resulting mRSA and RSA signatures are indistinguishable since: $w^{d_i^u} * w^{d_i^{sem}} = w^{d_i^u + d_i^{sem}} = w^{d_i} \bmod n$. Consequently, the mRSA signature generation process is transparent to eventual signature verifiers, since both the verification process and the signature format are identical to those in standard RSA.

Security. We briefly discuss the security of the signature scheme of Figure 2. At a high level, we require two properties: (1) the user cannot generate signatures after being revoked, and (2) the SEM cannot generate signatures on behalf of the user. For both properties we require existential unforgeability under a chosen message attack. Precise security models for this scheme (used in a slightly different context of multicative version of mRSA) can be found in [Bellare and Sandhu 2001] where a proof of security is given.

Randomized encodings. Note that, we assumed above that the encoding procedure $EC()$ is deterministic, as in EMSA-PKCS1-v1_5 [Labs 2002] encoding and Full Domain Hash (FDH) encoding [Bellare and Rogaway 1996]. If $EC()$ is a randomized encoding, such as EMSA-PSS [Labs 2002], we have to make sure both the user and SEM use the same randomness so that the resulting signature is valid. At the

same time, to prevent the user from generating signatures without its help, the SEM has to ensure that the random bits used for the encoding are chosen independently at random. Therefore, we cannot simply let the user choose the randomness for the encoding. Instead, the user and the SEM must engage in a two-party coin flipping protocol to generate the required shared randomness. Neither party can bias the resulting random bits. Consequently, these bits can be used as the randomness needed for the encoding function. However, when using deterministic encoding, such as EMSA-PKCS1-v1_5, there is no need for this step.

We note that in the above protocol the user sends the entire message m to the SEM in step (1). For privacy reasons, one might instead consider sending the digest $EC(m)$ the SEM. This would eliminate the difficulty with randomized encodings mentioned above. Unfortunately, the resulting system cannot be shown as secure as the underlying RSA signature scheme. Specifically, when only sending $EC(m)$ to SEM, we are unable to prove that the user cannot generate signatures after being revoked. The problem is that, while the user is not revoked, the SEM functions as an unrestricted RSA inversion oracle for the user. For example, the user can use the attack of [Desmedt and Odlyzko 1985] to generate signatures after being revoked. A proof of security is still possible, using a strong assumption on RSA: a variant of the "One-more RSA Assumption" [Bellare and Sandhu 2001]. Nevertheless, when using EMSA-PKCS1-v1_5 [Labs 2002] encoding, which is only heuristically secure, it might be fine to send $EC(m)$ to the SEM.

3.3 mRSA Decryption

Recall that PKCS1 [Labs 2002] stipulates that an input message m must be OAEP-encoded before carrying out the actual RSA encryption. We use $EC_{oaep}()$ and $DC_{oaep}()$ to denote OAEP encoding and decoding functions, respectively. The encryption process is identical to standard RSA, where $c = EC_{oaep}(m)^{e_i} \bmod n_i$ for each client U_i. Decryption, on the other hand, is very similar to mRSA signature generation described above.

1. U_i obtains encrypted message c and forwards it to its SEM.
   - SEM checks that U_i is not revoked and, if so, computes a partial clear-text $PC_{sem} = c^{d_i^{sem}} \bmod n_i$ and replies to the client.
   - concurrently, U_i computes $PC_u = c^{d_i^u} \bmod n_i$.
2. U_i receives PC_{sem} and computes $c' = PC_{sem} * PC_u \bmod n_i$. If OAEP decoding of c' succeeds, U_i outputs the clear-text $m = DC_{oaep}(c')$.

Security. We now briefly discuss the security of the mRSA decryption scheme shown in Figure 3. At a high level, we require two properties: (1) the user cannot decrypt ciphertexts encrypted with the user's public key after being revoked, and (2) the SEM cannot decrypt messages encrypted using the user's public key. For both properties we require semantic security under a chosen-ciphertext attack. Unfortunately, we cannot quite claim that the scheme above satisfies both properties.

OAEP and it variants are designed to provide chosen ciphertext security for RSA encryption in the random oracle model. The protocol above provides chosen ciphertext security against an attacker who is neither the SEM nor the user. However, OAEP does not necessarily satisfy properties (1) and (2) above. The problem is that the user can employ the SEM as an RSA inversion oracle until the user is

Algorithm: mRSA.decryption (executed by User and SEM)

(1) USER: $c \leftarrow$ encrypted message
(2) USER: send c to SEM
(3) In parallel:
  3.1. SEM:
  (a) If USER revoked return (ERROR)
  (b) $PC_{sem} \leftarrow c^{d_i^{sem}} \bmod n_i$
  (c) Send PC_{sem} to USER U_i
  3.2. USER
  (a) $PC_i^u \leftarrow c^{d_i^u} \bmod n_i$
(4) USER: $w \leftarrow (PC_{sem} * PC_u) \bmod n_i$
(5) USER: OAEP decode w. If success, output $m = DC_{oaep}(w)$.

Fig. 3. mRSA Decryption Algorithm

revoked. There is no way for the SEM to check whether a partial decryption it generates corresponds to a well-formed plaintext. However, as in the previous section, security can be proven in a weaker model under a strong assumption on RSA. (A detailed proof will be available in the extended version of this paper.)

We note that one way to make sure that the user cannot decrypt messages without the help of the SEM would be to use a Chosen Ciphertext Secure threshold cryptosystem [Shoup and Gennaro 1998; Canetti and Goldwasser 1999]. However, this would render the resulting scheme incompatible with currently deployed encryption systems (based on PKCS1).

3.4 Notable Features

As mentioned earlier, mRSA is only a slight modification of the RSA cryptosystem. However, at a higher level, mRSA affords some interesting features.

CA-based Key Generation. Recall that, in a normal PKI setting with RSA, a private/public key-pair is always generated by its intended owner. In mRSA, a client's key-pair is instead generated by a CA or some other trusted entity. Nonetheless, a CA only generates client's keys and does not need to keep them. In fact, a CA must erase them to assure that any successful future attack on the CA does not result in client's keys being compromised. In spite of that, having a trusted entity that generates private keys for a multitude of clients can be viewed as a liability. If CA-based key generation is undesirable then one can use a protocol of [Boneh and Franklin 2001] to distributively generate an RSA key between the SEM and the user. The downside is that this requires more work than letting the CA generate the key and give shares to the user and SEM. We note that CA-based key generation enables key escrow (provided that clients' keys are not erased after their out-of-band distribution). For example, if Alice is fired, her organization can still access Alice's encrypted work-related data by obtaining her private key from the CA.

Fast Revocation. The main point of mRSA is that the revocation problem is greatly simplified. In order to revoke a client's public key, it suffices to notify that client's SEM. Each SEM merely maintains a list of revoked clients which is consulted upon every service request. Our implementation uses standard X.509 Certificate

Revocation Lists (CRL's) for this purpose.

Transparency. mRSA is completely transparent to entities encrypting data for mRSA clients and those verifying signatures produced by mRSA clients. To them, mRSA appears indistinguishable from standard RSA. Furthermore, mRSA certificates are identical to standard RSA certificates. Thus, the SEM architecture is completely backwards compatible for the signature verifier and message encryptor.

Coexistence. mRSA's built-in revocation approach can co-exist with the traditional, explicit revocation approaches. For example, a CRL- or a CRT-based scheme can be used in conjunction with mRSA in order to accommodate existing implementations that require verifiers (and encryptors) to perform certificate revocation checks.

CA Communication. in mRSA, a CA remains an off-line entity. mRSA certificates, along with private half-keys are distributed to the client and SEM-s in an off-line manner. This follows the common certificate issuance and distribution paradigm. In fact, in our implementation (Section 6) there is no need for the CA and the SEM to ever communicate directly.

SEM Communication. mRSA does not require explicit authentication between a SEM and its clients. A client implicitly authenticates a SEM by verifying its own signature (or decryption) as described in Sections 3.2 and 3.3. These signature and encryption verification steps assure the client of the validity of SEM's replies. Although authentication of a client to a SEM is not required for the security of mRSA itself, it is needed for protection against denial-of-service attacks on a SEM. This can be easily accomplished with the authentication protocol described in Section 5.

Semi-trusted SEM. The SEM cannot issue messages on behalf of unrevoked users nor can it decrypt messages intended for unrevoked users. The worst-case damage caused by a compromise at the SEM is that users who were previously revoked can become unrevoked. This is similar to a compromise at a standard Revocation Authority which would enable the attacker to unrevoke revoked users.

4. ARCHITECTURE

The overall architecture is made up of three components: CA, SEM, and clients. A typical organizational setup involves one CA, a small set of SEM-s and a multitude of clients. A CA governs a small number of SEM-s. Each SEM, in turn, serves many clients. (In Section 5 we show how a single client can be supported by multiple SEM-s.) The assignment of clients to SEM-s is assumed to be handled off-line by a security administrator.

The CA component is a simple add-on to the existing CA and is thus still considered an off-line entity. For each client, the CA component takes care of generating an RSA public key, the corresponding certificate and a pair of half-keys (one for the client and one for the SEM) which, when combined, form the RSA private key. The respective half-keys are then delivered, out-of-band, to the interested parties.

The client component consists of the client library that provides the mRSA signature and mRSA decryption operations. It also handles the installation of the client's credentials at the local host.

The SEM component is the critical part of the architecture. Since a single SEM serves many clients, performance, fault-tolerance and physical security are of paramount concern. The SEM component is basically a daemon process that processes requests from its constituent clients. For each request, it consults its revocation list and refuses to help sign (or decrypt) for any revoked client. A SEM can be configured to operate in a stateful or stateless model. The former involves storing per client state (half-key and certificate) while, in the latter, no per client state is kept, however, some extra processing is incurred for each client request. The tradeoff is fairly clear: per client state and fast request handling versus no state and somewhat slower request handling.

4.1 Details

We now describe the SEM interaction in more detail. A client's request is initially handled by the SEM controller which checks the format of the request packet. Next, the request is passed on to the client manager which performs a revocation check. If the requesting client is not revoked, the request is handled depending on the SEM state model. If the SEM is stateless, it expects to find the so-called SEM *bundle* in the request. This bundle, as discussed in more detail later, contains the mRSA half-key, d_i^{SEM}, encrypted (for the SEM, using its public key) and signed (by the CA). The bundle also contains the RSA public key certificate for the requesting client. Once the bundle is verified, the request is handled by either the $\mathsf{mRSA}_{\mathsf{sign}}$ or $\mathsf{mRSA}_{\mathsf{decrypt}}$ component. If the SEM maintains client state, the bundle is expected only in the initial request. The same process as above is followed, however, the SEM's half-key and the client's certificate are stored locally. In subsequent client requests, the bundle (if present) is ignored and local state is used instead.

The security administrator communicates with a SEM via the administrative interface. This interface allows the administrator to manipulate the revocation list which, in our implementation is a regular X.509 CRL. (The X.509 format is not a requirement; a CRL can be represented in any signed format as long as it contains a list of revoked clients' certificate serial numbers.)

4.2 Implications of SEM Compromise

Suppose that an attacker compromises a SEM and learns d_i^{sem}. This knowlege can be used to "unrevoke" already revoked clients or block (ignore) future revocations. In the worst case, an attacker could neutralize SEM's mandatory involvement and thus cause the system to degrade to the reliance on normal revocation techniques, such as CRLs. However, we stress that knowledge of d_i^{sem} alone does not enable an attacker to decrypt or sign messages on behalf of a client.

Another interesting side-effect is the observation that there is no need to revoke all clients public keys whose key shares are exposed due to a compromised SEM. As long as a given client is not malicious (or compromised) its public key can remain the same. Specifically, in case of SEM compromise, a CA can simply generate a new pair of mRSA private half-keys for a given client using the same RSA (e_i, d_i, n_i) setting, but with a different SEM. This is possible because there are many ways to randomly "split" a given private exponent d into two parts. More generally, since each SEM client has a distinct RSA setting, even if a number of malicious clients collude, there is no danger to other (non-malicious) clients.

5. EXTENSIONS

We now briefly discuss several simple extensions of mRSA: multi-SEM support, mRSA blind signatures, identity-based mRSA, and authentication of mRSA requests.

Multi-SEM Support

Since each SEM serves many clients, a SEM failure – whether due to malicious or accidental causes – prevents all of its clients from decrypting data and generating signatures. To avoid having a single point of failure, mRSA can be modified to allow a single client to use multiple SEM-s.

The easiest approach is to physically replicate a SEM. While this helps with assuring service availability with respect to accidental (non-malicious) failures, replication does not protect against hostile attacks.

Another trivial solution is to allow a client to be served by multiple SEM-s, each with a different mRSA setting. This would require the CA to run the mRSA key generation algorithm t times (if t is the number of SEM-s) for each client. In addition to the increased computational load for the CA, this approach would entail each client having t distinct certificates or a single certificate with t public keys. The main disadvantage would be for other users (be they SEM clients or not) who would have to be aware of, and maintain, t public keys for a given SEM client.

Our approach allows a SEM client to have a single public key and a single certificate while offering the flexibility of obtaining service from any of a set of SEM-s. At the same time, each SEM maintains a different mRSA half-key for a given client. Thus, if any number of SEM-s(who support a given client) collude, they are unable to impersonate that client, i.e., unable to compute the client's half-key. Multi-SEM support involves making a slight change to the mRSA key generation algorithm, as shown in Figure 5.

Algorithm: mRSA.multi-key (executed by CA)

Choose a collision-resistant hash function $\mathtt{H} : \{0,1\}^* \rightarrow [1,\ldots,L]$ where $L \geq 1024$. Let k (even) be the security parameter. Assume client U_i is authorized to obtain service from $\{\mathsf{SEM}_0, \mathsf{SEM}_1, \ldots, \mathsf{SEM}_m\}$.

(1) Generate random $k/2$-bit primes: p, q
(2) $n_i \leftarrow p_i q_i$
(3) $e_i \stackrel{r}{\leftarrow} Z^*_{\phi(n_i)}$
(4) $d_i \leftarrow 1/e \bmod \phi(n_i)$
(5) $x \stackrel{r}{\leftarrow} Z_{n_i} - \{0\}$
(6) For each $j \in [0,\ldots,m]$, construct a server bundle for SEM_j:
$d_i^{sem} \leftarrow d_i - \mathtt{H}(x, SEM_j) \bmod \phi(n)$
(7) $SK_i \leftarrow (n_i, x)$
(8) $PK_i \leftarrow (n_i, e_i)$

Fig. 4. mRSA Key Generation for multiple SEM-s

To co-operate with SEM_j, the client simply computes $\mathtt{H}(x, SEM_j)$ as the corresponding mRSA half-key for the decryption or signatures.

Blind Signatures with mRSA

The concept of blind signatures was introduced by Chaum in [Chaum 1983] and [Chaum 1985]. They were originally intended for use in e-cash schemes where a bank (the actual signer) helps a client generate a signature on a message m, without knowledge of either m or the final signature.

mRSA can be easily extended to produce blind signatures. Suppose U_i wants to generate a blind signature on a message m. U_i first masks m by choosing $r \in_{\mathcal{R}} \mathbb{Z}^*_{n_i}$ and setting $m' = r^{e_i} EC(h(m)) \bmod n_i$. Then U_i sends a signature request on m' to SEM and, in the meantime, computes $PS_u = m'^{d_i^u} \bmod n_i$. SEM operates in the same way as in normal mRSA. When U_i receives PS_{sem}, it simply obtains $PS = r^{-1} * PS_u * PS_{sem} = EC(h(m))^{d_i} \bmod n_i$.

Identity-based mRSA

The concept of identity-based cryptosystems was introduced by Shamir in [Shamir 1985]. In an identity-based cryptosystem, all clients in an organization share a common cryptographic setting and their public keys are efficiently derived from their identities using a public algorithm. Therefore, personal public key certificates are not needed, which greatly simplifies certificate management and reduces reliance on PKI-s. Several identity-based signature systems have been developed in the past, e.g., [Guillou and Quisquater 1988]. The first practical identity-based encryption system was recently proposed by Boneh and Franklin in [Boneh and Franklin 2003]. However, no RSA-compatible identity-based cryptosystem has been developed thus far.

It turns out that mRSA can be modified to obtain an identity-based RSA variant (for both encryption and signatures) where clients share a common RSA modulus n and a client's public key e_i is derived from its identity. We briefly outline it here, however, a more detailed description can be found in [Ding and Tsudik 2003].

In this variant, only the CA knows the factorization of the common modulus n. For each client U_i, CA computes a unique public key e_i from the U_i's identity (e.g., its email addresses) using a collision-resistant hash function. Then, a CA computes the corresponding $d_i = e_i^{-1} \bmod \phi(n)$. The private key splitting as well as the signature and decryption are all the same as in mRSA, except that a CA does not issue an individual public key certificates to each client. Instead, a CA issues a site-wide (or organization-wide) attribute certificate, which includes, among other things, the common modulus n.

It is well-known that sharing a common modulus among multiple clients in plain RSA is utterly insecure, since knowledge of a single RSA public/private key-pair can be used to factor the modulus and obtain others' private keys. However, this is not an issue in identity-based mRSA since no client possesses an entire private key. However, collusion of a SEM and a single malicious client will result in a compromise of all clients of that SEM. Thus, a SEM in identity-based mRSA must be a fully trusted entity.

Authenticated mRSA

As discussed earlier, authentication of mRSA client requests can provide protection against denial-of-service (DoS) attacks on a SEM. To address DoS attacks, we

can modify both mRSA signature and decryption protocols to allow the SEM to authenticate incoming requests. For example, a client U_i can use its half-key d_i^u to sign each SEM -bound request message. (This can be done, for example, by having the client generate a partial signature on each request that is then verified by the SEM, much in the same manner that a client verifies SEM's reply in Step 5 of Figure 2.)

Although this method is simple and requires no additional set up costs, it does not really prevent DoS attacks, since a SEM would need to perform two modular exponentiations to authenticate each request. A simpler, more cost-effective approach is to use a MAC or keyed hash, e.g., HMAC [Bellare et al. 1997], to authenticate client requests. Of course, this would require a shared secret key between a SEM and each client. A CA could help in the generation and distribution of such shared secrets at the time of mRSA key generation. Yet another alternative is to rely on more general encapsulation techniques, such as SSL, to provide a secure channel for communication between SEM-s and clients.

6. IMPLEMENTATION

We implemented the entire SEM architecture for the purposes of experimentation and validation. The reference implementation is publicly available at `http://sconce.ics.uci.edu/sucses`. Following the SEM architecture described earlier, the implementation is composed of three parts:

(1) CA and Admin Utilities:
includes certificate issuance and revocation interface.

(2) SEM daemon:
SEM architecture as described in Section 4

(3) Client libraries:
mRSA client functions accessible via an API.

The reference implementation uses the popular OpenSSL library as the low-level cryptographic platform. OpenSSL incorporates a multitude of cryptographic functions and large-number arithmetic primitives. In addition to being efficient and available on many common hardware and software platforms, OpenSSL adheres to the common PKCS standards and is in the public domain.

The SEM daemon and the CA/Admin utilities are implemented on Linux and Unix while the client libraries are available on both Linux and Windows platforms.

In the initialization phase, CA utilities are used to set up the RSA public keypair for each client (client). The set up process follows the description in Section 3. Once the mRSA parameters are generated, two structures are exported: 1) server or SEM *bundle*, which includes the SEM's half-key d_i^{SEM}, and 2) client *bundle*, which includes d_i^u, the new certificate, and the entire server bundle if SEM is a stateless server.

A SEM bundle is a PKCS7 envelope. It contains d_i^{SEM} encrypted with the SEM's public key and signed by the CA. The client bundle is in PKCS12 format integrating the password privacy and public key integrity modes: it is signed by the CA and encrypted with the client-supplied key which can be derived from a password or a passphrase. (Note that a client cannot be assumed to have a pre-existing public key.)

.

After issuance, the client bundle is distributed out-of-band to the appropriate client. Before attempting any mRSA transactions, the client must first decrypt the bundle with its key and verify the CA's signature. Finally, the client's new certificate, the SEM bundle and half-key are extracted and stored locally.

To sign or decrypt a message, the client starts with sending a mRSA request with the SEM bundle piggybacked. The SEM processes the request and the bundle contained therein as described in Section 4. (Recall that the SEM bundle is processed based on the state model of the particular SEM.) If SEM can successfully open its bundle, it checks the whether the client's certificate is on the revocation list. If not, SEM follows the protocol and returns a corresponding reply to the client.

6.1 Email client plug-in

To further demonstrate the ease of use and practicality of the SEM architecture, we implemented plug-ins for both Eudora email reader and Outlook 2000 email client. When sending signed email, the plug-in reads the client bundle described in the previous section. It obtains the SEM address from the bundle and then communicates with the SEM to sign the email. The resulting signed email can be verified using any S/MIME capable email client such as Microsoft Outlook. In other words, the email recipient is oblivious to the fact that a SEM is used to control the sender's signing capabilities. When reading an encrypted email, the plug-in automatically loads the client bundle and decrypts the message by cooperating with SEM. For the sender, all S/MIME capable email composers can encrypt email for mRSA clients without any changes.

6.2 mRSA Email Proxy

An alternative way to use mRSA is through an mRSA-enabled email proxy. A proxy resides on the client's local host, runs in the background (as a daemon on Unix or a TSR program on Windows) and relays email messages between the local host and a remote SMTP server. An outbound email message, if requested, can be processed by the mRSA proxy using the same mRSA protocol as in the plug-in. For inbound email, the proxy can decrypt or verify signatures, if necessary. The main benefit of using a proxy is that it provides a single unified interface to the end-client and all email applications. This obviates the need to customize or modify email clients and offers a greater degree of transparency as well as ease of installation and configuration.

7. EXPERIMENTAL RESULTS

We conducted a number of experiments in order to evaluate the efficiency of the SEM architecture and our implementation.

We ran the SEM daemon on a Linux PC equipped with an 800 MHz Pentium III processor. Two different clients were used. The fast client was on another Linux PC with a 930 MHz Pentium III. Both SEM and fast client PC-s had 256M of RAM. The slow client was on a Linux PC with 466 MHz Pentium II and 128M of RAM. Although an 800 MHz processor is not exactly state-of-the-art, we opted to err on the side of safety and assume a relatively conservative (i.e., slow) SEM platform. In practice, a SEM might reside on much faster hardware and is likely to be assisted by an RSA hardware acceleration card.

Keysize (bits)	Data Size (bytes)	Comm. latency (ms)
512	102	4.0
1024	167	4.5
2048	296	5.5

Table I. Communication latency

Each experiment involved one thousand iterations. All reported timings are in milliseconds (rounded to the nearest 0.1 ms). The SEM and client PCs were located in different sites interconnected by a high-speed regional network. All protocol messages are transmitted over UDP.

Client RSA key (modulus) sizes were varied among 512, 1024 and 2048 bits. (Though it is clear that 512 is not a realistic RSA key size any longer.) The timings are only for the mRSA sign operation since mRSA decrypt is operationally almost identical.

7.1 Communication Overhead

In order to gain precise understanding of our results, we first provide separate measurements for communication latency in mRSA. Recall that both mRSA operations involve a request from a client followed by a reply from a SEM. As mentioned above, the test PCs were connected by a high-speed regional network. We measured communication latency by varying the key size which directly influences message sizes. The results are shown in Table I (message sizes are in bytes). Latency is calculated as the round-trip delay between the client and the SEM. The numbers are identical for both client types.

7.2 Standard RSA

As a point of comparison, we initially timed the standard RSA sign operation in OpenSSL (Version 0.9.6) with three different key sizes on each of our three test PCs. The results are shown in Tables II and III. Each timing includes a message hash computation followed by an modular exponentiation. Table II reflects optimized RSA computation where the *Chinese Remainder Theorem (CRT)* is used to speed up exponentiation (essentially exponentiations are done modulo the prime factors rather than modulo N). Table III reflects unoptimized RSA computation without the benefit of the CRT. Taking advantage of the CRT requires knowledge of the factors (p and q) of the modulus n. Recall that, in mRSA, neither the SEM nor the client know the factorization of the modulus, hence, with regard to its computation cost, mRSA is more akin to unoptimized RSA.

As evident from the two tables, the optimized RSA performs a factor of 3 to 3.5 faster for the 1024- and 2048-bit moduli than the unoptimized version. For 512-bit keys, the difference is slightly less marked.

7.3 mRSA Measurements

The mRSA results are obtained by measuring the time starting with the message hash computation by the client (client) and ending with the verification of the

Key-size (bits)	466 MHz PII (slow client)	800 MHz PIII (SEM)	930 MHz PIII (fast client)
512	2.9	1.4	1.4
1024	14.3	7.7	7.2
2048	85.7	49.4	42.8

Table II. Standard RSA (with CRT) signature timings in ms.

Key-size (bits)	466 MHz PII (slow client)	800 MHz PIII (SEM)	930 MHz PIII (fast client)
512	6.9	4.0	3.4
1024	43.1	24.8	21.2
2048	297.7	169.2	144.7

Table III. Standard RSA (without CRT) signature timings in ms.

Key-size (bits)	466 MHz PII (slow client)	930 MHz PIII (fast client)
512	8.0	9.9
1024	45.6	31.2
2048	335.6	178.3

Table IV. mRSA signature timings in ms.

signature by the client. The measurements are illustrated in Table IV.

It comes as no surprise that the numbers for the slow client in Table IV are very close to the unoptimized RSA measurements in Table III. This is because the time for an mRSA operation is determined solely by the client for 1024- and 2048- bit keys. With a 512-bit key, the slow client is fast enough to compute its PS_u in $6.9ms$. This is still under $8.0ms$ (the sum of $4ms$ round-trip delay and $4ms$ RSA operation at the SEM).

The situation is very different with a fast client. Here, for all key sizes, the timing is determined by the sum of the round-trip client-SEM packet delay and the service time at the SEM. For instance, $178.3ms$ (clocked for 2048-bit keys) is very close to $174.7ms$ which is the sum of $5.5ms$ communication delay and $169.2ms$ unoptimized RSA operation at the SEM.

All of the above measurements were taken with the SEM operating in a stateful mode. In a stateless mode, SEM incurs further overhead due to the processing of the SEM bundle for each incoming request. This includes decryption of the bundle and verification of the CA's signature found inside. To get an idea of the mRSA overhead with a stateless SEM, we conclude the experiments with Table V showing the bundle processing overhead. Only 1024- and 2048-bit SEM key size was considered. (512-bit keys are certainly inappropriate for a SEM.) The CA key size was constant at 1024 bits.

SEM key size	Bundle overhead
1024	8.1
2048	50.3

Table V. Bundle overhead in mRSA with a SEM in a stateless mode (in milliseconds).

8. RELATED WORK

Our system is constructed on top of a 2-out-of-2 threshold RSA algorithm, for the purpose of instant certificate revocation. In the following, we compare it with others' work with related functionality as well as those with similar cryptographic setting.

8.1 Current Revocation Techniques

Certificate revocation is a well-recognized problem in all current PKI-s. Several proposals attempt to address this problem. We briefly review these proposals and compare them to the SEM architecture. For each, we describe how it applies to signatures and encryption. We refer to the entity validating and revoking certificates as the Validation Authority (VA). Typically, a VA and a CA are one and the same. However, in some cases (such as OCSP) these are separate entities.

CRLs and Δ-CRLs: these are the most common ways to handle certificate revocation. The Validation Authority (VA) periodically posts a signed list (or another data structure) containing all revoked certificates. Such lists are placed on designated servers, called CRL Distribution Points. Since a list can get quite long, a VA may post a signed Δ-CRL which only contains the list of certificates revoked since the last CRL was issued. In the context of encrypted email, at the time email is sent, the sender checks if the receiver's certificate is included in the latest CRL. To verify a signature on a signed email message, the verifier first checks if (at present time) the signer's certificate is included in the latest CRL.

OCSP: the Online Certificate Status Protocol (OCSP) [Myers et al. 1999] avoids the generation and distribution of potentially long CRLs and provides more timely revocation information. To validate a certificate in OCSP, the client sends a certificate status request to the VA. The VA sends back a signed response indicating the status (revoked, valid, unknown) of the specified certificate.

We remark that the current OCSP protocol prevents one from implementing binding signature semantics: it is impossible to ask an OCSP responder whether a certificate was valid at some time in the past. Hopefully, this will be corrected in future versions of OCSP. One could potentially abuse the OCSP protocol and provide binding semantics as follows. To sign a message, the signer generates the signature, and also sends an OCSP query to the VA. The VA responds with a signed message saying that the certificate is currently valid. The signer appends both the signature and the response from the VA to the message. To verify the signature, the verifier checks the VA's signature on the validation response. The response from the VA provides a proof that the signer's certificate is currently valid. This method reduces the load on the VA: it is not necessary to contact the VA every

time a signature is verified. Unfortunately, there is currently no infrastructure to support this mechanism.

Certificate Revocation Trees: Kocher suggested an improvement over OCSP [Kocher 1998]. Since the VA is a global service it must be sufficiently replicated in order to handle the load of all the validation queries. This means the VA's signature key must be replicated across many servers which is either insecure or expensive (VA servers typically use tamper-resistance to protect the VA's signing key). Kocher's idea is to have a single highly secure VA periodically post a signed CRL-like data structure to many insecure VA servers. Users then query these insecure VA servers. The data structure proposed by Kocher is a hash tree where the leaves are the currently revoked certificates sorted by serial number (lowest serial number is the left most leaf and the highest serial number is the right most leaf). The root of the hash tree is signed by the VA. This hash tree data structure is called a Certificate Revocation Tree (CRT).

When a client wishes to validate a certificate CERT she issues a query to the closest VA server. Any insecure VA can produce a convincing proof that CERT is (or is not) on the CRT. If n certificates are currently revoked, the length of the proof is $O(\log n)$. In contrast, the length of the validity proof in OCSP is $O(1)$.

Skip-lists and 2-3 trees: One problem with CRT-s is that, each time a certificate is revoked, the whole CRT must be recomputed and distributed in its entirety to all VA servers. A data structure allowing for dynamic updates would solve this problem since a secure VA would only need to send small updates to the data structure along with a signature on the new root of the structure. Both 2-3 trees proposed by Naor and Nissim [Naor and Nissim 2000] and skip-lists proposed by Goodrich [Goodrich et al. 2001] are natural and efficient for this purpose. Additional data structures were proposed in [Aiello et al. 1998]. When a total of n certificates are already revoked and k new certificates must be revoked during the current time period, the size of the update message to the VA servers is $O(k \log n)$ (as opposed to $O(n)$ with CRT's). The proof of certificate's validity is $O(\log n)$, same as with CRTs.

A note on timestamping. Binding signature semantics (Section 2.2) for signature verification states that a signature is considered valid if the key used to generate the signature was valid at the time of signature generation. Consequently, a verifier must establish exactly when a signature was generated. Hence, when signing a message, the signer must interact with a trusted timestamping service to obtain a trusted timestamp and a signature over the client's (signed) message. This proves to any verifier that a signature was generated at a specific time. All the techniques discussed above require a signature to contain a trusted timestamp indicating when a signature was issued. There is no need for a trusted time service to implement binding signature semantics with the SEM architecture. This is because a SEM can be used to provide a secure time-stamping service as part of its mandatory involvement in each client's signature.

8.2 Two-party RSA

Several other research results developed schemes similar to the SEM architecture although in different security domains. Among them, Yaksha [Ganesan 1996] and S-RSA [MacKenzie and Reiter 2001b; 2001a] are the schemes conceptually closest

to ours. Both Yaksha and S-RSA involve 2-party RSA function sharing where clients do not possess complete RSA private keys and rely on an on-line server to perform certain private key operations.

The Yaksha system is a reusable security infrastructure which includes key exchange and key escrow. It allows a legitimate authority to recover clients' short-term session keys without knowing their long-term private keys. The client and the Yaksha server separately hold two shares such that their product forms a complete RSA private key for the client. When a Yaksha server receives a request for generating a session key, it chooses the key at random, encrypts it with the client's public key and decrypts it partially with the corresponding key share so that the result can be decrypted by the client using the other share.

Compared with our scheme, Yaksha is more expensive. A Yaksha client is unable to perform its local computation before it receives the server's result, whereas, the client's and SEM's computations in our scheme are executed concurrently. Also, a Yaksha server is a fully trusted entity and its compromise completely breaks the system security. In contrast, a SEM is only partially trusted; its compromise can only impair the intended service. Furthermore, a Yaksha server is a single point of failure and not scalable in that it serves for all users in the system . Our scheme allows multiple SEM-s, each serving (possibly overlapping) subsets of clients.

Another related result, S-RSA, is due to MacKenzie and Reiter [MacKenzie and Reiter 2001b; 2001a]. It aims to safeguard password-protected private keys on a captured networked device from offline dictionary attacks. In this scheme, the client's share is derived from its password; the server's share is contained in a token encrypted with the server's public key and stored in the device. The sum of the two shares forms the client's private RSA key. When needed, the encrypted token is sent to the server which extracts the key share and helps the client to issue a signature. The client is also able to notify the server to disable its key share by revealing certain secret information. Although the underlying cryptographic algorithms are similar to ours, the goals are fundamentally different: we focus on fine-grained control and fast revocation while S-RSA aims to protect networked devices.

Many other two-party schemes have been proposed in the literature. For example, Boneh and Franklin [Boneh and Franklin 2001] showed how to share the RSA key generation function between two parties. Nicolosi, et al. [Nicolosi et al. 2003] designed a proactive two-party Schnorr signature scheme. MacKenzie and Reiter [MacKenzie and Reiter 2001c] developed a provable secure two-party DSA signature scheme. However, none of these schemes are used in the content of revocation of security privileges.

9. SUMMARY

We described a new approach to certificate revocation and fine-grained control over security capabilities. Rather than revoking the client's certificate our approach revokes the client's ability to perform cryptographic operations such as signature generation and decryption. This approach has several advantages over traditional certificate revocation techniques: (1) revocation is fast – when its certificate is revoked, the client can no longer decrypt or sign messages, (2) with binding signature

semantics, there is no need to validate the signer's certificate as part of signature verification, and (3) our revocation technique is transparent to the peers since it uses standard RSA signature and encryption formats.

We implemented the SEM architecture for experimentation purposes. Our measurements show that signature and decryption times are not significantly higher from the client's perspective. Therefore, we believe the SEM architecture is appropriate for small- to medium-sized organizations where tight control of security capabilities is desired. The SEM architecture is clearly not appropriate for the global Internet or for educational campus-like environments.

REFERENCES

Aiello, W., Lodha, S., and Ostrovsky, R. 1998. Fast digital identity revocation. In *Advances in Cryptology – CRYPTO '98*, H. Krawczyk, Ed. Number 1462 in Lecture Notes in Computer Science. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany.

Bellare, M., Canetti, R., and Krawczyk, H. 1997. HMAC: Keyed-hashing for message authentication. Internet Request for Comment RFC 2104, Internet Engineering Task Force. Feb.

Bellare, M. and Rogaway, P. 1996. The exact security of digital signatures: How to sign with rsa and rabin. In *Advances in Cryptology – EUROCRYPT '96*, U. Maurer, Ed. Number 1070 in Lecture Notes in Computer Science. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany.

Bellare, M. and Sandhu, R. 2001. The security of practical two-party rsa signature schemes.

Boneh, D. and Franklin, M. 2001. Efficient generation of shared RSA keys. *Journal of the ACM (JACM) 48,* 4, 702–722. Extended abstract in Crypto '97.

Boneh, D. and Franklin, M. 2003. Identity-based encryption from the Weil Pairing. *SIAM J. of Computing 32,* 3, 586–615. Extended abstract in Crypto '01.

Boyd, C. 1989. Digital multisignatures. *Cryptography and Coding*, 241–246.

Canetti, R. and Goldwasser, S. 1999. An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack. In *Advances in Cryptology – EUROCRYPT '99*, J. Stern, Ed. Number 1592 in Lecture Notes in Computer Science. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany.

Chaum, D. 1983. Blind signatures for untraceable payments. In *Advances in Cryptology – CRYPTO '82*, R. L. Rivest, A. Sherman, and D. Chaum, Eds. Plenum Press, New York, 199–203.

Chaum, D. L. 1985. Security without identification: transaction systems to make big brother obsolete. *Commun. ACM 28,* 10 (Oct.), 1030–1044.

Desmedt, Y. and Odlyzko, A. 1985. A chosen text attack on the rsa cryptosystem and some discrete logarithm schemes. In *Advances in Cryptology – CRYPTO '85.*

Ding, X. and Tsudik, G. 2003. Simple identity-based encryption with mediated RSA. In *Progress in Cryptology - CT-RSA 2003.* LNCS 2612. Springer-Verlag, Berlin Germany.

Ganesan, R. 1996. The yaksha security system. *Commun. ACM 39,* 3 (Mar.), 55–60.

Gemmel, P. 1997. An introduction to threshold cryptography. *RSA CryptoBytes 2,* 7.

Goodrich, M., Tamassia, R., and Schwerin, A. 2001. Implementation of an authenticated dictionary with skip lists and commutative hashing. In *Proceedings of DARPA DISCEX II.*

Guillou, L. and Quisquater, J. 1988. Efficient digital public-key signature with shadow. In *Advances in Cryptology – CRYPTO '87*, C. Pomerance, Ed. Number 293 in Lecture Notes in Computer Science. International Association for Cryptologic Research, Springer-Verlag, Santa Barbara, CA, USA, 223–223. Lecture Notes in Computer Science No. 293.

Kocher, P. 1998. On certificate revocation and validation. In *Financial Cryptography – FC '98, Lecture Notes in Computer Science, Springer-Verlag, Vol. 1465.* 172–177.

Labs, R. 2002. PKCS #1v2.1: RSA cryptography standard. Tech. rep., RSA Laboratories. June.

MacKenzie, P. and Reiter, M. 2001a. Delegation of cryptographic servers for capture-resilient devices. In *8th ACM Conference on Computer and Communications Security*, P. Samarati, Ed. ACM Press, Philadelphia, PA, USA.

MacKenzie, P. and Reiter, M. 2001b. Networked cryptographic devices resilient to capture. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*. IEEE Computer Society, Technical Committee on Security and Privacy, IEEE Computer Society Press, Oakland, CA, 12–25.

MacKenzie, P. and Reiter, M. 2001c. Two-party generation of DSA signatures. In *Advances in Cryptology – CRYPTO '2001*, J. Kilian, Ed. Number 2139 in Lecture Notes in Computer Science. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 137–154.

McDaniel, P. and Rubin, A. 2000. A Response to 'Can We Eliminate Certificate Revocation Lists?'. In *Proceedings of the 4rd Conference on Financial Cryptography (FC '00)*, Y. Frankel, Ed. Number 1962 in Lecture Notes in Computer Science. International Financial Cryptography Association (IFCA), Springer-Verlag, Berlin Germany, Anguilla, British West Indies.

Myers, M., Ankney, R., Malpani, A., Galperin, S., and Adams, C. 1999. RFC 2560: Internet public key infrastructure online certificate status protocol - OCSP.

Naor, M. and Nissim, K. 2000. Certificate revocation and certificate update. *IEEE Journal on Selected Areas in Communications 18,* 4 (Apr.), 561–570.

Neuman, C. and Ts'o, T. 1994. Kerberos: An authentication service for computer networks. *IEEE Computer 32,* 9 (September).

Nicolosi, A., Krohn, M., Dodis, Y., and Mazières, D. 2003. Proactive two-party signatures for user authentication. In *Symposium on Network and Distributed Systems Security (NDSS '03)*. Internet Society, San Diego, CA.

Shamir, A. 1985. Identity-based cryptosystems and signature schemes. In *Advances in Cryptology – CRYPTO '84*, G. Blakley and D. Chaum, Eds. Number 196 in Lecture Notes in Computer Science. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 47–53.

Shoup, V. and Gennaro, R. 1998. Securing threshold cryptosystems against chosen ciphertext attack. In *Advances in Cryptology – EUROCRYPT '98*, K. Nyberg, Ed. Number 1403 in Lecture Notes in Computer Science. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 1–16.

Simple Identity-Based Cryptography with Mediated RSA

Xuhua Ding and Gene Tsudik

Department of Information and Computer Science, University of California, Irvine.
Email: {xhding,gts}@ics.uci.edu

Abstract. Identity-based public key encryption facilitates easy introduction of public key cryptography by allowing an entity's public key to be derived from an arbitrary identification value, such as name or email address. The main practical benefit of identity-based cryptography is in greatly reducing the need for, and reliance on, public key certificates. Although some interesting identity-based techniques have been developed in the past, none are compatible with popular public key encryption algorithms (such as El Gamal and RSA). This limits the utility of identity-based cryptography as a transitional step to full-blown public key cryptography. Furthermore, it is fundamentally difficult to reconcile fine-grained revocation with identity-based cryptography.
Mediated RSA (mRSA) [9] is a simple and practical method of splitting a RSA private key between the user and a Security Mediator (SEM). Neither the user nor the SEM can cheat one another since each cryptographic operation (signature or decryption) involves both parties. mRSA allows fast and fine-grained control of users' security privileges. However, mRSA still relies on conventional public key certificates to store and communicate public keys. In this paper, we present IB-mRSA, a simple variant of mRSA that combines identity-based and mediated cryptography. Under the random oracle model, IB-mRSA with OAEP[7] is shown as secure (against adaptive chosen ciphertext attack) as standard RSA with OAEP. Furthermore, IB-mRSA is simple, practical, and compatible with current public key infrastructures.
Keywords: Identity-based Encryption, Mediated RSA, Revocation

1 Introduction

In a typical public key infrastructure (PKI) setting, a user's public key is explicitly encoded in a public key certificate which is, essentially, a binding between the certificate holder's identity and the claimed public key. This common model requires universal trust in certificate issuers (Certification Authorities or CAs). It has some well-known and bothersome side-effects such as the need for cross-domain trust and certificate revocation. The main problem, however, is the basic assumption that all certificates are public, ubiquitous and, hence, readily available to anyone. We observe that this assumption is not always realistic, especially, in wireless (or any fault-prone) networks where connectivity is sporadic.

In contrast, identity-based cryptography changes the nature of obtaining public keys by constructing a one-to-one mapping between identities and public keys. Identity-based cryptography thus greatly reduces the need for, and reliance on, public key certificates and certification authorities. In general, identity-based encryption and identity-based signatures are useful cryptographic tools that facilitate easy introduction of, and/or conversion to, public key cryptography by allowing a public key to be derived from arbitrary identification values such as email addresses or phone numbers. At the same time, identity-based methods greatly simplify key management since they reduce both: the need for, and, the number of, public key certificates.

The concept of identity-based public encryption was first proposed by Shamir[20] in 1984. For the following 16 years the progress in this area has been rather slow. However, recently, Boneh and Franklin developed an elegant Identity-Based Encryption system (BF-IBE) based on Weil Pairing on elliptic curves [10]. BF-IBE represents a significant advance in cryptography.

Nevertheless, an identity-based RSA variant has remained elusive for the simple reason that an RSA modulus n (a product of two large primes) can not be safely shared among multiple users. Another notable drawback of current identity-based cryptographic methods is lack of support for fine-grained revocation. Revocation is typically done via Certificate Revocation Lists (CRLs) or similar structures. However, IBE aims to simplify certificate management by deriving public keys from identities, which makes it difficult to control users' security privileges.

In this paper, we propose a simple identity-based cryptosystem developed atop some Mediated RSA (mRSA) by Boneh, et al. [9]. mRSA is a practical and RSA-compatible method of splitting an RSA private key between the user and the security mediator, called a SEM. Neither the user nor the SEM knows the factorization of the RSA modulus and neither can decrypt/sign message without the other's help. By virtue of requiring the user to contact its SEM for each decryption and/or signature operation, mRSA provides fast and fine-grained revocation of users' security privileges.

Built on top of mRSA, IB-mRSA blends the features of identity-based and mediated cryptography and also offers some practical benefits.[1] Like mRSA, it is fully compatible with plain RSA. With the exception of the identity-to-public-key mapping, it requires no special software for communicating parties. IB-mRSA also allows optional public key certificates which facilitates easy transition to a conventional PKI. More generally, IB-mRSA can be viewed as a simple and practical technique inter-operable with common modern PKIs. At the same time, IB-mRSA offers security comparable to that of RSA, provided that a SEM is not compromised. Specifically, it can be shown that, in the random oracle model, IB-mRSA with OAEP[7] is as secure – against adaptive chosen ciphertext attacks – as RSA with OAEP.

The rest of the paper is organized as follows. The next section gives a detailed description of IB-mRSA. The security analysis is presented in Section 3 and the performance analysis – in Section 4. In Section 5, IB-mRSA is compared with Boneh-Franklin's IBE. Finally, a brief description of the implementation is presented in the last section. Some further security details can be found in the Appendix.

[1] A very sketchy version of IB-mRSA was first presented in [8].

2 Identity-Based mRSA

The main feature of identity-based encryption is the sender's ability to encrypt messages using the public key derived from the receiver's identity and other public information. The identity can be the receiver's email address, user id or any value unique to the receiver; essentially, an arbitrary string. To compute the encryption key, an efficient (and public) mapping function $\mathcal{KG}$ must be set beforehand. This function must be a one-to-one mapping from identity strings to public keys.

The basic idea behind identity-based mRSA is the use of a single common RSA modulus n for all users within a system (or domain). This modulus is public and contained in a system-wide certificate issued, as usual, by some Certificate Authority (CA). To encrypt a message for a certain recipient (Bob), the sender (Alice) first computes $e_{Bob} = \mathcal{KG}(ID_{Bob})$ where ID_{Bob} is the recipient's identity value, such as Bob's email address. Thereafter, the pair (e_{Bob}, n) is treated as a plain RSA public key and normal RSA encryption is performed. On Bob's side, the decryption process is identical to that of mRSA.

We stress that using the same modulus by multiple users in a normal RSA setting is **utterly insecure**. It is subject to a trivial attack whereby anyone – utilizing one's knowledge of a single key-pair – can simply factor the modulus and compute the other user's private key. However, in the present context, we make an important assumption that: *Throughout the lifetime of the system, the adversary is unable to compromise a SEM.*

Obviously, without this assumption, IB-mRSA would offer no security whatsoever: a single SEM break-in coupled with the compromise of just one user's key share would result in the compromise of all users' (for that SEM) private keys. The IB-mRSA assumption is slightly stronger than its mRSA counterpart. Recall that, in mRSA, each user has a different RSA setting, i.e., a unique modulus. Therefore, to compromise a given user an adversary has to break into both the user and its SEM.

We now turn to the detailed description of the IB-mRSA scheme.

2.1 System Setting and User Key Generation

In the following, we use email addresses as unique identifiers of the public key owners in the system. However, as mentioned above, other identity types can be used just as well, e.g., Unix UIDs, HTTP addresses, physical addresses or even phone numbers. We use the notation ID_{Alice} to denote the user's (Alice) email address that will be used to derive the public exponent.

In the initialization phase, a trusted party (CA) sets up the RSA modulus for all users in the same system (organization or domain). First, CA chooses, at random, two large primes p' and q' such that $p = 2p' + 1$ and $q = 2q' + 1$ are also primes, and finally sets $n = pq$. We note that, since n is a product of two strong primes, a randomly chosen odd number in Z_n has negligible probability of not being relatively prime to $\phi(n)$. (See Section 3 for further discussion.) Hence, the mapping function $\mathcal{KG}$ can be quite trivial. (Our current implementation uses the popular $MD5$ hash function.)

The public exponent e_{Alice} is constructed as the output of $\mathcal{KG}(ID_Alice)$ represented as a binary string of the same length as the modulus, with the least significant bit

set. This ensures that e_{Alice} is odd and, with overwhelming probability, relatively prime to $\phi(n)$. The complete IB-mRSA key generation proceeds as in Figure 1.

Algorithm IB-mRSA.key (executed by CA)
Let k (even) be the security parameter

1. Generate random $k/2$-bit primes: p', q' s.t. $p = 2p' + 1, q = 2q' + 1$ are also prime.
2. $n \leftarrow pq, e \in_{\mathcal{R}} \mathbb{Z}^*_{\phi(n)}, d \leftarrow e^{-1} \bmod \phi(n)$
3. For each user (Alice):
   (a) $s \leftarrow k - |\mathcal{KG}()| - 1$
   (b) $e_{Alice} \leftarrow 0^s || \mathcal{KG}(ID_A) || 1$
   (c) $d_{Alice} \leftarrow 1/e_{Alice} \bmod \phi(n)$
   (d) $d_{Alice,u} \stackrel{r}{\leftarrow} Z_n - \{0\}$
   (e) $d_{Alice,sem} \leftarrow (d - d_{Alice,u}) \bmod \phi(n)$

Fig. 1. IB-mRSA: User Key Generation

A domain- or system-wide certificate ($Cert_{org}$) is issued by the CA after completion of the key generation algorithm. This certificate contains almost all the usual fields normally found in RSA public key certificates with few exceptions, such as no real public key value is given. In particular, it mainly contains the common modulus n and (if applicable) the common part of the email address for all users, such as the domain name.

For the sake of compatibility with other (not identity-based) RSA implementations – including plain RSA and mRSA – the CA may, upon request, issue an individual certificate to a user. In most cases, however, an individual user certificate would not be needed, since not having such certificates is exactly the purpose of identity-based cryptography.

2.2 IB-mRSA Encryption

To encrypt a message, the sender needs only the recipient's email address and the domain certificate. The encryption algorithm is shown in Figure 2.

Algorithm IB-mRSA.encr

1. Retrieve n, k and $\mathcal{KG}$ algorithm identifier from the domain certificate;
2. $s \leftarrow k - |\mathcal{KG}()| - 1$
3. $e \leftarrow 0^s || \mathcal{KG}(ID_A) || 1$
4. Encrypt input message m with (e, n) using standard RSA/OAEP, as specified in PKCS#1v2.1[3]

Fig. 2. IB-mRSA: Encryption

Since the receiver's public key is derived from the receiver's unique identifier, the sender does not need a public key certificate to ensure that the intended receiver is the correct public key holder. Furthermore, fast revocation provided by mRSA obviates the need for the sender to perform any revocation checks. The decryption process is essentially the same as in mRSA. If a certain user needs to be revoked, the domain security administrator merely notifies the appropriate SEM and the revoked user is unable to decrypt any further messages.

2.3 IB-mRSA Decryption

IB-mRSA decryption is identical to that of mRSA. To make this paper self-contained, we borrow (from [9]) the protocol description in Figure 3. For a detailed description and security analysis of additive mRSA, we refer the reader to [9].[2]

Protocol IB-mRSA.decr (executed by User and SEM)

1. USER: $m' \leftarrow$ encrypted message
2. USER: send m' to SEM
3. In parallel:
   3.1 SEM:
   (a) If USER revoked return (ERROR)
   (b) $PD_{sem} \leftarrow m'^{d_{sem}} \bmod n$
   (c) Send PD_{sem} to USER
   3.2 USER:
   (a) $PD_u \leftarrow m'^{d_u} \bmod n$
4. USER: $M \leftarrow (PD_{sem} * PD_u) \bmod n$
5. USER: $m \leftarrow$ OAEP Decoding of M
6. USER: If succeed, return (m)

Fig. 3. IB-mRSA: Decryption

3 Security of Identity-based mRSA

We now examine the security of IB-mRSA/OAEP in a setting with n users. All users share a common RSA modulus N and each user (U_i) is associated with a unique identity ID_i, which is mapped into an RSA public exponent e_i via a mapping function $\mathcal{KG}$.

3.1 Security Analysis

In the following, we argue that if $\mathcal{KG}$ is an appropriate hash function, IB-mRSA/OAEP is semantically secure against adaptive chosen ciphertext attacks (CCA-2) in the random oracle model. We use the term *indistinguishability* which is a notion equivalent to semantic security. (See [6] for the relevant discussion.)

[2] There is also a very similar multiplicative mRSA (*mRSA) first proposed by Ganesan [13].

Our analysis is mainly derived from the results in [5], ([4] has similar results) where it was shown that a public-key encryption system in a multi-user setting is semantically secure against certain types of attacks if and only if the same system in a single-user setting is semantically secure against the same attack types.

IB-mRSA/OAEP is obviously an encryption setting with many users, although they do not physically possess their own private keys. To prove semantic security, we begin by asserting that IB-mRSA in single-user mode is equivalent to the standard RSA/OAEP, which is proven secure against CCA-2 under the random oracle [12]. Next, we apply the theorems in [5] with the condition that all users are honest. To remove this condition, we analyze the distribution of views of the system from users and outside adversaries. Furthermore we introduce an additional requirement for the key generation function (division-intractability) so that we can neglect the possibility of an attack from legitimate (inside) users, which is a problem unique to our setting. In the end, we argue for semantic security of IB-mRSA/OAEP.

We use $Succ_1^{IB}(t, q_d)$ to denote the maximum advantage of all adversary algorithms in polynomial time t, attacking IB-mRSA/OAEP with one user, $Succ_n^{IB}(t, q_d, q_e)$ for the setting with n users, and $Succ^R(t, q_d)$ for RSA/OAEP. In the above, q_d(q_e) denote the maximum number of decryption (encryption) queries allowed for each public key. Throughout the analysis, we consider semantic security against CCA-2 under the random oracle assumption. To conserve space, we omit mentioning them in the following discussion.

We begin with the following lemma.

Lemma 1. *IB-mRSA/OAEP system in a single-user setting is polynomially as secure as standard RSA/OAEP encryption, i.e.,*

$$Succ_1^{IB}(t, q_d) = Succ^R(t', q_d)$$

where c is constant value, $t' = t + c$.

The proof is in Appendix A.2. Basically, if there exists an algorithm breaking the security of IB-mRSA/OAEP in a single-user mode, we can build upon it an algorithm breaking standard RSA/OAEP with the same success probability and constant extra overhead. Of course, it is easy to see that breaking RSA/OAEP implies breaking IB-mRSA. Thus, we claim that they are equally secure.

For the multi-user setting, we cannot claim that IB-mRSA with n users is semantically secure by directly applying the security reduction theorem in [5]. The reason is that our system is not a typical case referred in [5]. Sharing a common RSA modulus among many users results in their respective trapdoors not being independent; consequently, there could be attacks among the users. Furthermore, users in IB-mRSA may have the incentive not only to attack other users, but also to attempt to break the underlying protocol so that they can bypass the mandatory security control of the SEM.

However, assuming for the moment, that all users are honest, we can obtain the following lemma derived from [5].

Lemma 2. *IB-mRSA/OAEP system with n users is semantically secure if all n are honest. More precisely,*

$$Succ_n^{IB}(t_n, q_d, q_e) \leq q_e n Succ_1^{IB}(t_1, q_d)$$

where $t_1 = t_n + O(\log(q_e n))$

When all users are honest, there are clearly no attacks. Thus, IB-mRSA with multi-user can be considered as an example of encryption system in [5] where each user has an independent trapdoor. We adapt the original proof in [5] in order to claim security against CCA-2 since no user actually knows its own trapdoor in IB-mRSA. See Appendix A.3 for details.

Unfortunately, in a real application, all users cannot be assumed to be trusted. To remove this condition in Lemma 2, we have to examine both the information an inside user can observe and the operations an inside user can perform.

For a given entity (user or set of users) we use an informal term "system view" to refer to the distribution of all inputs, outputs, local state information as well as scripts of interactions with decryption oracles, encryption oracles, and the SEM. The system view for an outside attacker is denoted as:

$$V_1 ::= Pr\{N, (e_0, \ldots e_n), \Gamma_O, \Gamma_E, \Gamma_D, \Gamma_{SEM}\}$$

while the system view for a set of users is:

$$V_2 ::= Pr\{N, (e_0, \ldots, e_n), \{d_{u_i}\}, \Gamma_O, \Gamma_E, \Gamma_D, \Gamma_{SEM}, \Gamma_{d_u,n}\}$$

where $\{d_{u_i}\}$ is the set of user key-shares; $\Gamma_O, \Gamma_E, \Gamma_D$ are three scripts recording all queries/answers to the random oracle, encryption oracles and decryption oracles, respectively; Γ_{SEM} is the script recording all requests/replies between all users and the SEM; $\Gamma_{d_u,n}$ is the script recording all n users' computation on ciphertexts with their own secret key-share d_{u_i}. We claim in Lemma 3, that being an IB-mRSA user does not afford one extra useful information as compared to an outside adversary.

Lemma 3. *Under the adaptive chosen ciphertext attack, the system view of the outside adversary (V_1), is polynomially indistinguishable from the combined system view (V_2) of a set of malicious insiders, in the random oracle model.*

Proof. See Appendix A.4 for details. □

Thus far, we have shown that insider adversaries do not gain advantages over outsiders in terms of obtaining extra information. However, we also need to consider the privileged operations that an insider can make. In IB-mRSA, each user is allowed to send legitimate decryption queries to its SEM. In conventional proofs, the adversary is not allowed to query its oracle for the challenge ciphertext. However in our case, an inside adversary can manipulate a challenge ciphertext (intended for decryption with d_i) into another ciphertext that can be decrypted with its own key d_j and legally decrypt it with the aid of the SEM.[3]

[3] A simple example is as follows. Suppose $e_i = 3 * e_j$. Then, given $c = m^{e_j} \bmod n$, User U_i can compute $c' = c^3 = m^{e_i} \bmod n$, which can be decrypted by U_i with the help from its SEM. The notion of non-malleability does not capture this attack since it is defined under a single fixed private/public key pair.

We now have to consider the probability of such attacks in our setting. More generally, let $e_{a_0}, \ldots, e_{a_v}$ be the set of public keys of v malicious users, and $E_v = \prod_{a_i} e_{a_i}$. They may attempt to use some function f, which takes a challenge $c = m^x \bmod n$ as input and outputs ciphertext $c' = m^{E_v}$. We offer the following lemma to address the conditions for the existence of such f.

Lemma 4. *Given two RSA exponents x, y and modulus n, let f be a polynomial time complexity function s.t. $f(m^x) = m^y \bmod n$. Such f exists iff $x|y$.*

Proof. See Appendix A.5 for details. □

According to Lemma 4, we require negligible probability of obtaining a user's public key which is a factor of the product of a set of others. A similar requirement appears in a signature scheme by Gennaro et al. in [14]. They introduce the notion of division intractability for a hash function. Informally, a hash function H is **Division intractable** if it is infeasible to find distinct $(X_1, \ldots, X_n, Y)$ in its domain, such that $H(Y)|\prod_i(H(X_i))$. Denoting $Pr^{div}(H)$ as the probability that H fails to hold this property, we have the following proposition regarding the security of IB-mRSA in a multi-user setting.

Proposition 1. *IB-mRSA/OAEP encryption offers equivalent semantic security to RSA/OAEP against adaptive chosen ciphertext attacks in the random oracle model, if the key generation function is division intractable.*

In summary, Lemma 3 and Lemma 4 enable us to remove the condition of Lemma 2 where all users are assumed to be honest, by requiring the key generation function to be division intractable. Thus, we can reduce the security of IB-mRSA/OAEP in multi-user setting into single-user, which is as secure as standard RSA/OAEP according to Lemma 1.

3.2 The Public Key Mapping Function

The key generation function $\mathcal{KG}$ in IB-mRSA is a hash function H. To ensure the security of the scheme, H must satisfy the following requirements.

AVAILABILITY OF PUBLIC KEYS: The output of H should have an overwhelming probability of being relatively prime to $\phi(n)$. Obviously, for the inverse (private key) to exist, a public exponent can not have common factors with $\phi(n)$.

Note that in Section 2 the RSA modulus n is set to $n = p * q$ and p, q are chosen as strong primes $p = 2p' + 1$, $q = 2q' + 1$ where both p' and q' are also large primes. Considering $\phi(n) = 2^2 p'q'$ with only three factors 2, p, q, the probability of the output from H being co-prime to $\phi(n)$ is overwhelming on the condition that the output is an odd number, because finding an odder number not co-prime to $4p'q'$ is equivalent to find p' or q' and consequently factoring n.

COLLISION RESISTANCE: H should be a collision-resistant function, i.e., given any two distinct inputs ID_1, ID_2, the probability of $H(ID_1) = H(ID_2)$ should be negligible. In other words, no two users in the domain can share the same public exponent.

DIVISION RESISTANCE: As discussed in Section 3.1, division intractability of H is essential to the security of IB-mRSA. Gennaro et al. analyzed the probability of division for hash functions in [14].

Moreover, Coron and Naccache showed in [11] that the number of necessary hash-value to find a division relation among a hash function's outputs is sub-exponential to its digest size k: $exp(\sqrt{2\log 2}/2 + o(1)\sqrt{k \log k})$.

They suggested using 1024-bit hash functions to get a security level equivalent to 1024-bits RSA. However, such a strong hash function is not needed in our case. As a point of comparison, the GHR signature scheme [14] needs a division-intractable hash function to compute message digests, where an adaptive adversary can select any number of inputs to the underlying hash function. IB-mRSA needs a hash function to compute digests from users' identities. In any domain, the number of allowed identities is certainly much fewer compared to the number of messages in [14].

digest size in bits	$\log_2$ complexity (in # of operations)
128	36
160	39
192	42
1024	86

Table 1. Estimated complexity of the attack for variable digest sizes.

To help select the best hash size for our purposes, we quote from the experiments by Coron and Naccache [11] in Table 1. Taking the first line as an example, an interpretation of the data is that, among at least 2^{36} hash digests, the probability of finding one hash value dividing another is non-negligible. In IB-mRSA setting, the typical personnel of an organization is on the order of $2^{10} \sim 2^{17}$. Consequently, the possible number of operations is far less than 2^{36}. Hence, we can safely use MD5 or SHA-1 as the mapping function (H).

3.3 SEM Security

Suppose that the attacker is able to compromise the SEM and expose the secret key d_{sem}, however, without collusion with any user. This only enables the attacker to "un-revoke" previously revoked, or block possible future revocation of currently valid, certificates. The knowledge of d_{sem} does not enable the attacker to decrypt or sign messages on behalf of the users. The reason is obvious: note that Alice never sends her partial results to her SEM. Thus, the attacker's view of Alice can be simulated in the normal RSA setting, where the attacker just picks a random number as d_{sem} and make computations on the ciphertext, messages to sign and signatures generated by Alice.

3.4 Security of Common Modulus

As mentioned earlier, using a common RSA modulus is clearly unacceptable in plain RSA setting. In the mediated RSA architecture, sharing a modulus is feasible since no

party knows a complete private/public key-pair. In fact, no coalition of users is able to compute a public/private key-pair. The only way to "break" the system appears to be by subverting a SEM and colluding with a user. Thus, in the context of IB-mRSA we need to assume that a SEM is a **fully** trusted party, as opposed to **semi-trusted** in mRSA [9].

4 Performance Analysis

When plain RSA is used for encryption, the public encryption exponent e is typically a small integer with only a few 1-bits. One example is the popular OpenSSL toolkit [17] which uses $65,537$ as the default public key value for RSA certificates. Encryption with such small exponents can be accelerated with specialized algorithms for modular exponentiation. However, in IB-mRSA setting, there is no such luxury of choosing special exponents and a typical public exponent is a relatively large integer with (on the average) half of the bits set to 1.

Keys	RSA Modulus 1Kb	RSA Modulus 2Kb	RSA Modulus 4Kb
$65,537$	2 ms	4 ms	12 ms
128-bit key	7 ms	20 ms	69 ms
160-bit key	8 ms	25 ms	88 ms

Table 2. IB-mRSA Encryption: Performance Comparison of Different Encryption Keys.

We ran some simple tests to assess the cost of IB-mRSA encryption for public keys derived from email addresses. The encryption was tested using OpenSSL on an 800MHz PIII workstation. In the tests, we used: 1) "default" encryption exponent $65,537$ and 2) two other exponents of length 128-bit and 160-bit. For each key, we randomly set half of the bits. The results are depicted in Table 2.

From the results in Table 2, we see that encryption with a randomized key does introduce overhead, especially when the RSA modulus size grows. However, it is rather negligible for the 1024-bit case, which is currently the most popular modulus size.

The decryption cost for IB-mRSA is identical to mRSA. The performance of mRSA has been reported on by Boneh, et al. in [9]. For example, a 1024-bit mRSA decryption costs around 35ms on an 800 MHz PIII, as compared to 7.5ms for plain RSA on the same platform. We note that this is still much cheaper than 40ms that is needed for Boneh/Franklin IBE decryption (for 1024 bits of security on a even more powerful hardware platform).

5 IB-mRSA versus Boneh/Franklin IBE

We now provide a detailed comparison of BF-IBE and IB-mRSA. The comparison is done along several aspects, including: practicality, revocation, security and cost of key generation.

Practicality and Performance: Although BF-IBE and IB-mRSA have similar architectures, the underlying cryptographic primitives are completely different. Compared to the elliptic curve primitives used in BF-IBE, IB-mRSA is much easier to deploy since RSA is currently the most popular public key encryption method. Recall that IB-mRSA is fully compatible with standard RSA encryption. Moreover, if optional individual certificates are used, IB-mRSA is fully compatible with current PKI-s. Thus, it offers a smooth and natural transition from normal ID-based to public key cryptography.

In addition, IB-mRSA offers better performance than BF-IBE. As seen from the comparison in Table 3, IB-mRSA is noticeably faster than BF-IBE in both key generation and message encryption.

	BF-IBE	IB-mRSA
Private Key Generation	3ms	$<$ 1ms
Encryption Time	40ms	7ms
Decryption Time	40ms	35ms

Table 3. Performance Comparison of BF-IBE (on PIII 1GHz) and IB-mRSA (on PIII 800MHz) with 1024-bit security.

Revocation: BF-IBE does not explicitly provide revocation of users' security capabilities. This is natural since it aims to avoid the use of certificates in the course of public key encryption. On the other hand, revocation is often necessary and even imperative.

The only way to obtain revocation in normal IBE is to require fine-grained time-dependent public keys, e.g., public keys derived from identifiers combined with time- or date-stamps. This has an unfortunate consequence of having to periodically re-issue all private keys in the system. Moreover, these keys must be (again, periodically) securely distributed to individual users. In contrast, IB-mRSA inherits its fine-grained revocation functionality from mRSA [9]. IB-mRSA provides per-operation revocation, whereas, BF-IBE provides periodic revocation, which clearly has coarser granularity. Essentially, IB-mRSA allows revocation to commence at any time while BF-IBE revokes users by refusing to issue new private keys. However, BF-IBE does not prevent the type of an attack whereby an adversary who compromises a previous or current key can use them to decrypt previously encrypted messages. This can be a serious attack in some settings, such as military applications.

Trusted Third Parties: Both SEM in IB-mRSA and PKG in BF-IBE are trusted third parties. However, the difference in the degree of trust is subtle. A SEM is fully trusted since its collusion with any user can result in a compromise of all other users' secret keys, due to the shared RSA modulus. Nonetheless, a compromise of a SEM alone does not result in a compromise of any users' secret keys. A PKG is a real TTP since it knows all users' secrets, thus, a compromise of a PKG results in a total system break. While a PKG can also be a CA at the same time, a SEM can never be allowed to play the role of CA.

If BF-IBE is used to provide fine-grained revocation, frequent key generation and secure key distribution are expensive procedures. Although a PKG is not required to be on-line all of the time, in practice, it must be constantly available since users do not all request their current private keys at the same time. Therefore, as the revocation interval in BF-IBE gets smaller, the on-line presence of a PKG becomes more necessary.

6 Implementation

We implemented IB-mRSA for the purposes of experimentation and validation. The implementation is publicly available at `http://sconce.ics.uci.edu/sucses`. The software is composed of three parts:

1. CA and Admin Utilities: domain certificate, user key generation, (optional) certificate issuance and revocation interface.
2. SEM daemon: SEM process as described in Section 2
3. Client libraries: IB-mRSA user functions accessible via an API.

The code is built on top of the popular OpenSSL [17] library. OpenSSL incorporates a multitude of cryptographic functions and large-number arithmetic primitives. In addition to being efficient and available on many common hardware and software platforms, OpenSSL adheres to the common PKCS standards and is in the public domain. The SEM daemon and the CA/Admin utilities are implemented on Linux, while the client libraries are available on both Linux and Windows platforms.

In the initialization phase, a CA initializes the domain-wide cryptographic setting, namely (n, p, q, p', q') and selects a mapping function (currently defaulting to MD5) for all domain clients. The set up process follows the description in Section 2. For each user, two structures are exported: 1) SEM *bundle*, which includes the SEM's half-key d_i^{SEM}, and 2) user *bundle*, which includes d_i^u and the entire server bundle.

The server bundle is in PKCS#7[1] format, which is basically a RSA envelope signed by the CA and encrypted with the SEM's public key. The client bundle is in PKCS#12[2] format, which is a shared-key envelope also signed by the CA and encrypted with the user-supplied key which can be a pre-set key, a password or a passphrase. (A user is not assumed to have a pre-existing public key.)

After issuance, each user bundle is distributed in an out-of-band fashion to the appropriate user. Before attempting any IB-mRSA transactions, the user must first decrypt and verify the bundle. A separate utility program is provided for this purpose. With it, the bundle is decrypted with the user-supplied key, the CA's signature is verified, and, finally, the user's half-key are extracted and stored locally.

To decrypt a message, the user starts with sending an IB-mRSA request, with the SEM bundle piggybacked. The SEM first check the status of the client. Only when the client is deemed to be a legitimate user, does the SEM process the request using the bundle contained therein. As mentioned earlier, in order to encrypt a message for an IB-mRSA, that user's domain certificate needs to be obtained. Distribution and management of domain certificates is assumed to be done in a manner similar to that of normal certificate, e.g., via LDAP or DNS.

6.1 Emailer client plug-in

To demonstrate the ease of using IB-mRSA we implemented plug-ins for the popular Eudora[19] and Outlook[16] mailers. The plug-ins allow the sender to encrypt outgoing emails to any client in the common domain using only one domain (organizational) certificate. When ready to send, the sender's plug-in reads the recipient's email address and looks up the organization certificate by using the domain name in the email address. A screen snapshot of the Eudora plug-in is shown in Figure 4.

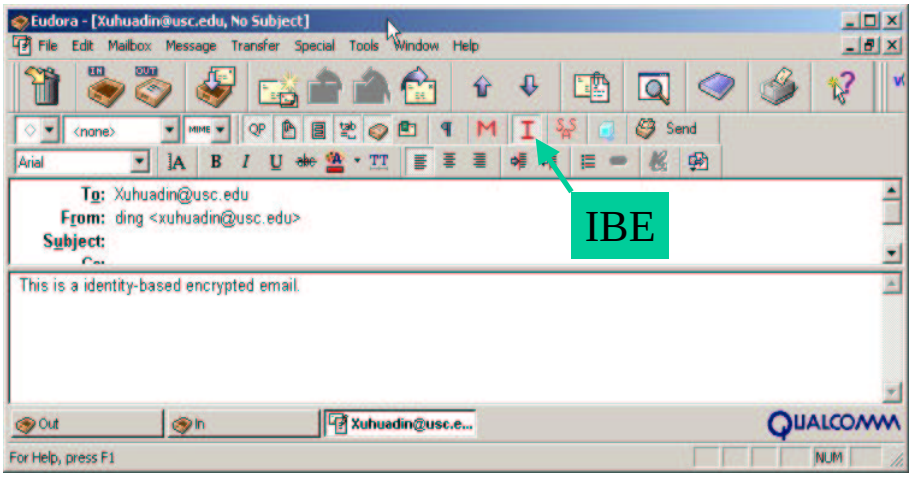

Fig. 4. Eudora IBE Plugin

When an email message encrypted with IB-mRSA is received, an icon for IB-mRSA is displayed in the message window. To decrypt the message, the user just clicks on the IB-mRSA icon. The plug-in then contacts the user's SEM to get a partially decrypted message (if the user is not revoked). This is basically the same process as in mRSA.

7 Summary and Future Work

We described IB-mRSA, a practical and secure identity-based encryption scheme. It is compatible with standard RSA encryption and offers fine-grained control (revocation) of users security privileges.

Several issues remain for future work. It is unclear whether IB-mRSA can be shown secure under the standard model (our argument utilizes the random oracle setting). Moreover, we need a more formal analysis of semantic security. Another issue relates to IB-mRSA performance. Using a hash function for public key mapping makes encryption more expensive than RSA since the public exponent is random (and on the average half of the bits are set). We need to investigate alternative mapping functions that can produce more "efficient" RSA exponents.

8 Acknowledgements

Some of the initial ideas for this work emanated from discussions with Dan Boneh whose contribution is gratefully acknowledged. Thanks also go to the anonymous reviewers for their useful comments.

References

1. PKCS #7: Cryptographic message syntax standard. Technical note, RSA Laboratories, Nov. 1993. Version 1.5.
2. PKCS #12: Personal information exchange syntax. Technical note, RSA Laboratories, 1999. Version 1.0.
3. PKCS#1v2.1: Rsa cryptography standard. Technical note, RSA Laboratories, 2002. Version 2.1.
4. O. Baudron, D. Pointcheval, and J. Stern. Extended notions of security for multicast public key cryptosystems. In *27th International Colloquium on Automata, Languages and Programming (ICALP '2000)*, number 1853 in Lecture Notes in Computer Science. Springer-Verlag, Berlin Germany, July 2000.
5. M. Bellare, A. Boldyreva, and S. Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In Preneel [18], pages 259–274.
6. M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In H. Krawczyk, editor, *Advances in Cryptology – CRYPTO '98*, number 1462 in Lecture Notes in Computer Science, pages 26–45. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 1998.
7. M. Bellare and P. Rogaway. Optimal asymmetric encryption — how to encrypt with RSA. In A. D. Santis, editor, *Advances in Cryptology – EUROCRYPT '94*, number 950 in Lecture Notes in Computer Science, pages 92–111. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 1995.
8. D. Boneh, X. Ding, and G. Tsudik. Identity based encryption using mediated rsa. In *3rd Workshop on Information Security Application*, Jeju Island, Korea, Aug. 2002. KIISC.
9. D. Boneh, X. Ding, G. Tsudik, and C. M. Wong. A method for fast revocation of public key certificates and security capabilities. In *10th USENIX Security Symposium*, Washington, D.C., Aug. 2001. USENIX.
10. D. Boneh and M. Franklin. Identity-based encryption from the Weil Pairing. In Kilian [15], pages 213–229.
11. J.-S. Coron and D. Naccache. Security analysis of the gennaro-halevi-rabin signature scheme. In Preneel [18], pages 91–101.
12. E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP is secure under the rsa assumption. In Kilian [15], pages 260–274.
13. R. Ganesan. Augmenting kerberos with pubic-key cryptography. In T. Mayfield, editor, *Symposium on Network and Distributed Systems Security*, San Diego, California, Feb. 1995. Internet Society.
14. R. Gennaro, S. Halevi, and T. Rabin. Secure hash-and-sign signatures without the random oracle. In J. Stern, editor, *Advances in Cryptology – EUROCRYPT '99*, number 1592 in Lecture Notes in Computer Science, pages 123–139. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 1999.
15. J. Kilian, editor. *Advances in Cryptology – CRYPTO '2001*, number 2139 in Lecture Notes in Computer Science. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 2001.

16. Microsoft. Microsoft Outlook©, http://www.microsoft.com.
17. OpenSSL User Group. The OpenSSL Project Web Page, http://www.openssl.org.
18. B. Preneel, editor. *Advances in Cryptology – EUROCRYPT '2000*, number 1807 in Lecture Notes in Computer Science, Brugge, Belgium, 2000. Springer-Verlag, Berlin Germany.
19. Qualcomm. Qualcomm eudora mailer, http://www.eudora.com.
20. A. Shamir. Identity-based cryptosystems and signature schemes. In G. Blakley and D. Chaum, editors, *Advances in Cryptology – CRYPTO '84*, number 196 in Lecture Notes in Computer Science, pages 47–53. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 1985.

A Proof of Security

A.1 Notations and Attack Model

Throughout the appendix, we use the following notations

- $\mathcal{KG}$: Key Generation Function
- $\mathcal{PE}()$: IB-mRSA/OAEP encryption system.
- $\mathcal{DO}_{d_i}$: Decryption oracle with private key d_i
- $\mathcal{RO}$:Random oracle
- N: The common RSA modulus
- e_i/d_i: the i-th user's public key/private key
- n: The number of users in $\mathcal{PE}$
- q_e:The number of encryptions allowed to be performed by each user
- q_d: The maximum number of decryption queries the adversary can ask

Under the notion of indistinguishability of security, the adversary $\mathcal{A}$ takes the public key and outputs two equal length messages m_0, m_1. Then, it gets a challenge ciphertext C, computed by an encryption oracle which secretly picks $b \in_{\mathcal{R}} \{0,1\}$ and encrypts m_b. $\mathcal{A}$ is challenged to output b with a probability non-negligibly greater than $1/2$. In CCA attack model, $\mathcal{A}$ is allowed to send queries to a decryption oracle, with the restriction that $\mathcal{A}$ is not allowed to query on the challenge ciphertext c.

A.2 Proof of Lemma 1

Proof. The lemma means that if there exists an attack algorithm $\mathcal{B}$ with polynomial time complexity, breaking the security of IB-RSA/OAEP with success probability ϵ, then there exists an attack algorithm $\mathcal{F}$ with the same polynomial degree of running time, breaking RSA/OAEP with the same success probability; and vice versa.

The reverse direction is obvious. For any $\mathcal{F}$ that can break the indistinguishability of standard RSA, it breaks IB-mRSA in single-user mode. Thus we have $Succ^R(t', q_d) \leq Succ_1^{IB}(t, q_d)$. Now we show $Succ_1^{IB}(t, q_d) \leq Succ^R(t', q_d)$.

Let $\mathcal{B}$ be the polynomial algorithm attacking on the indistinguishability of $\mathcal{PE}(\mathcal{KG}, N, 1)$ containing the single user U_0 and its public key e_0 and secret bundle d_{u_0}. By allowing $\mathcal{B}$ to know d_{u_0}, we model the concern that the user in the system may be malicious. We construct $\mathcal{F}$ as the adversary algorithm against the standard RSA/OAEP $(\hat{N}, \hat{e})$ and analyze its success probability and time complexity. Replacing $\mathcal{KG}$ function in $\mathcal{PE}$ by a random oracle and acting as the random oracle and decryption oracle for $\mathcal{B}$, $\mathcal{A}$ runs $\mathcal{F}$ as follows.

Experiment $\mathcal{F}^{RSA}(\hat{N}, \hat{e}, \mathcal{DO}_d, \mathcal{RO})$

Select two random messages (m_0, m_1) with equal length. The encryption oracle EO_e secretly selects a random bit $b \in_{\mathcal{R}} \{0, 1\}$ and encrypts m_b into the ciphertext c. Given c, $\mathcal{F}$ runs the following to determine b.

1. Generate a random number r and a string id;
2. Initialize $\mathcal{PE}(\mathcal{KG}, N)$ with single-user setting by $N \leftarrow \hat{N}$, For user $ID_0 \leftarrow id$;
3. $\mathcal{PE}$ queries its random oracle ($\mathcal{F}$) for e_0;
4. $e_0 \leftarrow \hat{e}$;
5. Initialize $\mathcal{B}$ with (m_0, m_1, c) and the target system $\mathcal{PE}(\mathcal{KG}, \hat{N})$ and user public key e_0, user bundle r;
6. Run $\mathcal{B}$. The number of decryption queries is bounded by q_d:
   (a) For all $\mathcal{B}$'s random oracle queries on OAEP encoding/decoding, $\mathcal{F}$ forwards them to $\mathcal{RO}$ and hands the answers back;
   (b) For all $\mathcal{B}$'s decryption oracle queries, $\mathcal{F}$ forwards them to $\mathcal{DO}_d$, and hands the answers back;
   (c) For $\mathcal{B}$'s requests c to SEM (remember that the adversary might be inside the system): $\mathcal{F}$ queries $\mathcal{DO}_d$ on c. On getting the reply $c^d \bmod n$, $\mathcal{F}$ hands back $c^d/c^r \bmod n$ as the reply from SEM to $\mathcal{B}$.
7. $\mathcal{B}$ halts outputting a bit b';
8. Return b';

Clearly, if $\mathcal{B}$'s output b' equals b, $\mathcal{F}$ successfully discovers b. This holds for all polynomial algorithm $\mathcal{B}$. Thus we have $Succ_1^{IB}(t, q_d) \leq Succ^R(t', q_d)$. As for the time complexity of $\mathcal{F}$, the steps 1∼5 and steps 7,8 take constant time, in that the cost is independent of the security parameter, and step 6 runs in time t . Hence, the overall time for $\mathcal{F}$ is $t + c$, which leads us to the conclusion of $Succ_1^{IB}(t, q_d) = Succ^R(t', q_d)$. □

A.3 Proof of Lemma 2

Proof. If all users in $\mathcal{PE}$ are considered trusted, we do not need to consider all attacks originated from the inside users. The $\mathcal{PE}$ is therefore a IB-mRSA/OAEP in multi-user setting, whose security for single-mode is proved in Lemma 1.

To show the polynomial reduction, the proof in [5] constructs an attack algorithm $\mathcal{B}$ for single-setting. $\mathcal{B}$ calls another algorithm $\mathcal{A}$, which can break the multi-user setting. In order to argue the security in CCA2 model, $\mathcal{B}$ has to simulate the decryption oracles for $\mathcal{A}$. This is simple in their case, where $\mathcal{B}$ can invoke key generation function to obtain all needed public/private key pairs. Unfortunately, this is not the case in IB-mRSA setting, since the key generation will not give $\mathcal{B}$ the private keys. We slightly revise the original proof.

Still, $\mathcal{A}$ targets at a multi-use setting with public keys $\{N, e_0, \ldots, e_n\}$. However, we construct $\mathcal{B}$, targeting at $\{N, e = \prod_{i=0}^{n}(e_i)\}$. The algorithm for $\mathcal{A}$'s decryption query is shown below:

Decryption oracle simulator ($\mathcal{B}, \mathcal{A}, N, e_0, \ldots, e_n, e = \prod_{i=0}^{n} e_i$)
$\mathcal{B}$ simulates decryption oracle for $\mathcal{A}$ with the help from its own oracle $\mathcal{DO}_d$.
(d is the corresponding secret key to (n, e).)

1. $\mathcal{A} \rightarrow \mathcal{B}$: (c, e_i),
2. $\mathcal{B} \rightarrow \mathcal{DO}_d$: c
3. $\mathcal{B} \leftarrow \mathcal{DO}_d$: $c' = c^d \bmod n$
4. $\mathcal{B}$ computes $b = \frac{e}{e_i}$
5. $\mathcal{A} \leftarrow \mathcal{B}$: $a = c'^b \bmod n$

One can easily check that the answer a is exactly c^{1/e_i}. Thus, the proof for Theorem 4.1 in [5] still holds, which also proves this lemma.
□

A.4 Proof of Lemma 3

Proof. (Sketch) No secret channel is assumed either in IB-mRSA protocol execution or in the attack model. Thus, the outsider observes everything that the insider does, except for $\{d_{u_i}\}$ and $\Gamma_{d_u,n}$. However, an outsider can simulate $\Gamma_{d_u,n}$ with the help of a random oracle and the decryption oracles.

Note that a user's key-share is nothing but a random number derived from an idealized hash function, which can be replaced by the random oracle RO. The outsider can query RO and obtain a set of random values $\{r_i\}$ with the same distribution as $\{d_{u_i}\}$. For each ciphertext c in $\Gamma_{d_u,n}$ (encrypted with e_{u_i}) the adversary constructs $\Gamma'_{d_u,n}$ by computing $c^{d_{u_i}} = c^{d_i}/c^{r_i}$, where c^{d_i} is obtained from the decryption oracle $\mathcal{DO}_{d_i}$. All c, d_{u_i},r_i are random integers. (Note that c is also random since OAEP encoding is applied before exponentiation). Thus, $Pr\{\Gamma_{d_u,n}\} = Pr\{\Gamma'_{d_u,n}\}$, which leads to V_1 and V_2 having the same distribution □

A.5 Proof of Lemma 4

Proof. We show that $x|y$ is a sufficient and necessary condition for the existence of f.
SUFFICIENCY: if $x|y$, i.e. $\exists k \in \mathcal{N}$, s.t. $y = kx$. We construct f as

$$f : a \rightarrow a^k \bmod n$$

One can easily check that f is the desired function.
NECESSITY: Suppose there exists a function f satisfying the requirement, while $y = kx + r$, where $k, r \in \mathcal{N}$ and $1 \leq r \leq x$. Given $c = m^x \bmod n$, we can compute $c_1 = f(c) = m^y \bmod n$. Suppose $g = gcd(x, y)$, i.e $\exists a, b \in \mathcal{Z}$, s.t. $ax + by = g$. Thus, in polynomial time, we can get m^g by computing $c^a c_1^b \bmod n$. If we let $x = hg$, we have actually constructed a polynomial-time algorithm, which, taking $c = (m^g)^h \bmod n$ and h, n as input, outputs $c^{1/h} \bmod n$ without knowing the factorization of n. (Note that x is relatively prime to $\phi(n)$, which implies that h is also relatively prime to $\phi(n)$ and is a valid RSA public key exponent.) However, this contradicts the RSA assumption. □

Identity-Based Mediated RSA

Dan Boneh[1], Xuhua Ding[2], and Gene Tsudik[2]

[1] Computer Science Department, Stanford University.
Email: dabo@cs.stanford.edu
[2] Department of Information and Computer Science, University of California, Irvine.
Email: {xhding,gts}@ics.uci.edu

Abstract. Identity-based encryption (IBE) [5] and digital signatures are important tools in modern secure communication. In general, identity-based cryptographic methods facilitate easy introduction of public key cryptography by allowing an entity's public key to be derived from some arbitrary identification value such as an email address or a phone number. Identity-based cryptography greatly reduces the need for, and reliance on, public key certificates.
Mediated RSA (mRSA) [4] is a simple and practical method of splitting RSA private keys between the user and the Security Mediator (SEM). Neither the user nor the SEM can cheat one another since each signature or decryption must involve both parties. mRSA allows fast and fine-grained control (revocation) over users' security privileges. However, mRSA still relies on public key certificates to derive public keys.
Current identity-based cryptographic methods do not support fine-grained revocation while mediated cryptography (such as mRSA) still relies on public key certificates to derive public keys. In this paper we present IB-mRSA, a variant of mRSA that combines identity-based and mediated cryptography. IB-mRSA is simple, secure and very efficient.

1 Introduction

In a typical public key setting, a user's public key is explicitly encoded in a public key certificate which is, essentially, a binding between the certificate holder's identity and the claimed public key. This common PKI model requires universal trust in certificate issuers (Certification Authorities or CAs). This also has some well-known side-effects such as cross-domain trust and certificate revocation. The main problem, however, is the basic assumption that all certificates are public, ubiquitous and, hence, readily available to anyone. We note that this assumption is not always realistic, especially, in wireless networks where connectivity is sporadic.

In contrast, identity-based cryptography changes the nature of obtaining public keys by constructing a one-to-one mapping between identities and public keys. Thus, identity-based cryptography greatly reduces the need for, and reliance on, public key certificates and certification authorities. Generally speaking, identity-based encryption and identity-based digital signatures are useful cryptographic tools that facilitate easy introduction of, and/or conversion to, public key cryptography by allowing a public key

to be derived from arbitrary identification values such as email addresses or phone numbers. At the same time, identity-based methods greatly simplify key management since they reduce both: the need for, and, the number of, public key certificates. However, one notable drawback is that current identity-based cryptographic methods do not support fine-grained revocation. (Revocation is typically done via Certificate Revocation Lists – CRLs or similar structures.)

Mediated RSA [4] is a simple and practical method of splitting RSA private keys between the user and the SEM. Neither the user nor the SEM can cheat one another since each signature or decryption operation must involve both parties. mRSA allows fast and fine-grained control (revocation) of users' security privileges. However, mRSA still relies on public key certificates to derive public keys.

Our goal in this paper is to combine the attractive features of identity-based cryptography with the fine-grained control of mRSA. To this end, we present IB-mRSA, a variant of mRSA that combines identity-based and mediated cryptography. IB-mRSA is simple, secure and very efficient.

Organization: The rest of this paper is organized as follows. In the next section we provide a brief overview of mediated RSA. Next, we describe IB-mRSA in Section 3 and analyze its security. Sections 5 and 6 discuss the implementation of IB-mRSA and its performance, respectively.

2 Overview of Mediated RSA

Mediated RSA (mRSA) involves a special entity, called a SEM an on-line partially trusted server. To sign or decrypt a message, Alice must first obtain a message-specific token from the SEM. Without this token Alice can not use her private key. To revoke Alice's ability to sign or decrypt, the administrator instructs the SEM to stop issuing tokens for Alice's public key. At that instant, Alice's signature and/or decryption capabilities are revoked. For scalability reasons, a single SEM serves many users. One of the mRSA's advantages is its transparency: SEM's presence is invisible to other users: in signature mode, mRSA yields standard RSA signatures, while in decryption mode, mRSA accepts plain RSA-encrypted messages.

The main idea behind mRSA is the splitting of an RSA private key into two parts as in threshold RSA [9]. One part is given to a user while the other is given to a SEM. If the user and the SEM cooperate, they employ their respective half-keys in a way that is functionally equivalent to (and indistinguishable from) standard RSA. The fact that the private key is not held in its entirety by any one party is transparent to the outside, i.e., to the those who use the corresponding public key. Also, knowledge of a half-key cannot be used to derive the entire private key. Therefore, neither the user nor the SEM can decrypt or sign a message without mutual consent.

We now provide a brief overview of mRSA functions. (For a detailed description and security analysis of mRSA, we refer the reader to [4].) The variant described below is the additive mRSA (+mRSA) as presented by Boneh, et al. in [4]. (There is also a very similar multiplicative mRSA (*mRSA) first proposed by Ganesan [8].) The first function `+mRSA.key` is used by the CA to set up the user's public/private key-pair and

split the private key into two shares. The other two, `+mRSA.sign` and `+mRSA.decr`, are the signing and decryption functions, respectively. We note that no encryption and signature verification functions are specified since they are identical to those in plain RSA.

`Algorithm +mRSA.key (executed by CA)`
`Let k (even) be the security parameter`

1. Generate random $k/2$-bit primes: p, q
2. $n \leftarrow pq$
3. $e \stackrel{r}{\leftarrow} Z^*_{\phi(n)}$
4. $d \leftarrow 1/e \bmod \phi(n)$
5. $d_u \stackrel{r}{\leftarrow} Z_n - \{0\}$
6. $d_{sem} \leftarrow (d - d_u) \bmod \phi(n)$
7. $SK \leftarrow d$
8. $PK \leftarrow (n, e)$

After computing the above values, CA securely communicates d_{sem} to the SEM and d_u – to the user. (See [4] for details.) The user's public key PK is released, as usual, in a public key certificate.

`Protocol +mRSA.sign (executed by User and SEM)`

1. USER: $h \leftarrow H(m)$
   where H() is a suitable hash function such as SHA-1 and $|H()| < k$
2. USER: send h to SEM.
3. In parallel:
   3.1 SEM:
   (a) If USER revoked return (ERROR)
   (b) $PS_{sem} \leftarrow h^{d_{sem}} \bmod n$
   (c) send PS_{sem} to USER
   3.2 USER:
   (a) $PS_u \leftarrow h^{d_u} \bmod n$
4. USER: $h' \leftarrow (PS_{sem} * PS_u)^e \bmod n$
5. USER: If $h' \neq h$ then return (ERROR)
6. $S \leftarrow (PS_{sem} * PS_u) \bmod n$
7. USER: return (h,S)

3 Identity-Based mRSA

The main feature of identity-based encryption is the sender's ability to encrypt messages using the public key derived from the receiver's identity and other public information.

Protocol +mRSA.decr (executed by User and SEM)

1. USER: $m' \leftarrow$ encrypted message
2. USER: send m' to SEM
3. In parallel:
   3.1 SEM:
   (a) If USER revoked return (ERROR)
   (b) $PD_{sem} \leftarrow m'^{d_{sem}} \bmod n$
   (c) Send PD_{sem} to USER
   3.2 USER:
   (a) $PD_u \leftarrow m'^{d_u} \bmod n$
4. USER: $m \leftarrow (PD_{sem} * PD_u) \bmod n$
5. USER: return (m)

The receiver's identity can be the receiver's email address, user id or any value unique to the receiver (essentially, an arbitrary string). To compute the receiver's encryption key, an efficient public mapping function $f()$ must be set beforehand. This function must be a one-to-one mapping from identity strings to public keys.

The basic idea behind identity-based mRSA is the use of a single common RSA modulus n among all users of a system. This modulus is assumed to be public and contained in a public key certificate issued, as usual, by some Certificate Authority (CA). To encrypt a message for a certain recipient (Bob), the sender (Alice) first computes $e_{Bob} = f(ID_{Bob})$ where ID_{Bob} is the recipient's identity value, such as Bob's email address. Thereafter, the pair (e_{Bob}, n) is treated as a plain RSA public key and normal RSA encryption is performed. On Bob's side, the decryption process is identical to that of mRSA.

We stress that using the same modulus by multiple users in a normal RSA setting is **utterly insecure**. It is subject to a trivial attack whereby anyone – utilizing one's knowledge of a single key-pair – can simply factor the modulus and compute the other user's private key. However, in our present context, we make an important assumption that:

> [IB-mRSA Assumption:] Throughout the lifetime of the system, the adversary is unable to compromise a SEM.

Obviously, without this assumption, IB-mRSA would offer no security: a single SEM break-in coupled with the compromise of just one user's key share would result in the compromise of all users' (for that SEM) private keys. The IB-mRSA assumption is slightly stronger that its mRSA counterpart. Recall that, in mRSA, each user has a different RSA setting, i.e., a unique modulus. Therefore, to compromise a given user an adversary has to break into both the user and its SEM.

We now turn to the detailed description of the IB-mRSA scheme.

3.1 System Setting and User Key Generation

In the following, we use email addresses as unique identifiers of the public key owners in the system. However, as mentioned above, other identity types can be used just as

well, e.g., Unix UIDs, HTTP addresses, physical addresses or even phone numbers. We use the notation ID_{Alice} to denote the user's (Alice) email address that will be used to compute the public exponent.

In the initialization phase, a trusted party (CA) sets up the RSA modulus for all users in the same system (organization or domain). First, CA chooses, at random, two large primes p' and q' such that $p = 2p' + 1$ and $q = 2q' + 1$ are also prime. Then it computes $n = pq$, which is, in fact, a Blum integer. Since n is a Blum integer, a randomly chosen number in Z_n has negligible probability of not being relatively prime to $\phi(n)$. (See Section 4 for further discussion.) Hence, our mapping function f can be quite trivial.

The public exponent is set to be the email address represented as a binary string, padded with zeros on the left and with the rightmost bit set to one. This ensures that e_{Alice} is odd and, with overwhelming probability, relatively prime to $\phi(n)$. It is assumed that the email address is at most 8 bits shorter than the size of the RSA modulus. This is reasonable considering practical RSA modulus sizes. The currently recommended minimum RSA modulus size is 1024 bits. Consequently, our assumption translates into the email address being at most 127 characters (bytes) long. We observe that most email addresses tend to be under 30 characters.

In fact, it is largely unnecessary to use the *entire* email address as input to $f()$; it suffices to use only its leftmost component, i.e., the "username" portion. This is because the rest of the address is common to all email users in the domain. For example, if Alice's email address is: `Alice.Smith@ics.uci.edu` we can use "`Alice.Smith`" as input to $f()$. The common part "`ics.uci.edu`" can be assumed to be part of the domain-wide (organizational) certificate.

The complete IB-mRSA key generation proceeds as follows:

`Algorithm IB-mRSA.key (executed by CA)`
`Let k (even) be the security parameter`

1. Generate random $k/2$-bit primes: p', q' s.t. $p = 2p' + 1, q = 2q' + 1$ are also prime.
2. $n \leftarrow pq$
3. For each user (Alice):
   (a) $k' \leftarrow k - |ID_Alice| - 8$
   (b) $e_{Alice} \leftarrow f(ID_{Alice}) = 0^{k'}||ID_{Alice}||00000001$
   (c) $d_{Alice} \leftarrow 1/e_{Alice} \bmod \phi(n)$
   (d) $d_{Alice,u} \stackrel{r}{\leftarrow} Z_n - \{0\}$
   (e) $d_{Alice,sem} \leftarrow (d - d_{Alice,u}) \bmod \phi(n)$

A domain- or system-wide certificate ($Cert_{org}$) is issued by the CA after completion of the key generation algorithm. This certificate contains all the usual fields normally found in RSA certificates with few exceptions discussed later in Section 4. In particular, it contains the common modulus n and (if applicable) the common part of

the email address for all users. User's and SEM's key shares are distributed securely as in the mRSA scheme described in Section 2.

For the sake of compatibility with other (not identity-based) RSA implementations – including plain RSA and mRSA – the CA may, upon request, issue an individual certificate to a user. In most cases, however, an individual user certificate would not be needed, since not having such certificates is exactly the purpose of identity-based cryptography.

3.2 IB-mRSA Encryption

To encrypt a message, the sender needs the recipient's email address and its organization certificate.

Algorithm IB-mRSA.encr

1. $e \leftarrow IB - mRSA.key(Email)$
2. Retrieve n from organization certificate;
3. Let (e, n) be the public key and encrypt message m using standard RSA encryption with OAEP padding.

Note that the recipient's public key certificate is not required for the sender to encrypt. Since the key is derived from the receiver's unique identifier, the sender does not need a certificate to ensure that the intended receiver is the correct public key holder. Furthermore, instantaneous revocation provided by mRSA obviates the need for the sender to perform any revocation checks. The decryption process is essentially the same as in mRSA: the security administrator merely notifies the SEM and the revoked user is unable to decrypt any further messages.

3.3 Signature Protocol

IB-mRSA can be also used for signing messages. Since the signing protocol is the same as in mRSA, we do not provide a detailed description. Basically, a user computes its own half-signature while the SEM computes its half-signature. When the two parts are coalesced together, a full-blown RSA signatures is obtained.

The verification procedure is slightly different from mRSA (or plain RSA). In mRSA and RSA, the verifier obtains the public verification key, from the signer's public key certificate. The certificate can be obtained from the signature structure itself or from some public site. In ID-based signature schemes, the verifier computes the signer's public key using the signer's identity. In IB-mRSA, as mentioned before, we still need a domain certificate, which includes the common RSA modulus n. However, we should point it out that this certificate is not a normal public key certificate but a sort of an attribute certificate for the entire domain. The verification protocol is as follows:

Note that, since signature generation is the same as in mRSA, the binding signature semantic is preserved [4]. This means that the verifier can be sure that the signer's

Algorithm IB-mRSA.verify

1. Extract and verify domain certificate from signature or from a public site.
2. Obtain RSA modulus n from domain certificate;
3. $e \leftarrow IB - mRSA.key(Email)$;
4. Let (e, n) be the public key and verify the signature using standard RSA verification algorithm with OAEP padding.

private key was valid (believed by the SEM to be valid) at the time the signature was computed.

4 Security Analysis

In this section, we discuss some security issues unique to IB-mRSA. Clearly, the biggest security concern for IB-mRSA is the use of the common modulus for all users within the same domain.

SEM ***Security:*** Let us consider an attacker trying to decrypt a message sent to Alice or to forge Alice's signature on a certain message. Recall that the token sent by SEM back to Alice is $t = x^{d_{sem}} \bmod N$ for some value of x. The attacker sees both x and the token t. In fact, since there is no authentication of the user's request to the SEM, the attacker can obtain this t for any x of its choice. We claim that this information is of no use to the attacker. The reason is that, d_{sem} is just a random number in Z_n independent of the rest of the attacker's view. Using a simulation argument, we claim that any attack possible with the SEM can be mounted to attack standard RSA. One can simulate the SEM by picking a random integer $d_{sem} \in_R Z_n$ and thus use the attack on the SEM to mount an attack on standard RSA.

Suppose the attacker is able to compromise the SEM and expose the secret key d_{sem}. This enables the attacker to "un-revoke" previously revoked, or block possible future revocation of currently valid, certificates. However, knowledge of d_{sem} does not enable the attacker to decrypt or sign messages on behalf of its users. The reason is obvious: Note that Alice does not send her partial results to her SEM. Thus the attacker's view of Alice can be simulated in the normal RSA setting, where the attacker just picks a random number as d_{sem} and make computations on the ciphertext, messages to sign and signatures generated by Alice.

Security of Common Modulus: As mentioned earlier, using a common RSA modulus is clearly unacceptable in plain RSA setting. In the mediated RSA architecture, sharing a modulus is feasible since no one knows a complete private/public key-pair. In fact, no coalition of users is able to compute a public/private key-pair. The only way to "break" the system appears to be by subverting a SEM. Thus, in the context of IB-mRSA we have to assume that a SEM is a **fully** trusted party, as opposed to **semi-trusted** in mRSA.

One unclear issue has to do with the existence of attacks on a common modulus with a large number of public/private key pairs. With one public/private key pair, it is widely believed that RSA encryption (with OAEP padding) is secure against adaptive chosen ciphertext attack [2], [7]. More specifically, such encryption is secure even if the attacker is allowed to employ a decryption oracle at will. In IB-mRSA we introduce a new attack model:

> The adversary is allowed to freely choose e', and ask the decryption/signature oracle to decrypt/sign a chosen ciphertext/plaintext with d' any number of times. (Where d' is the private counterpart of e' and RSA modulus is fixed.)

We refer to this type of a decryption oracle as a *multi-key oracle*. This oracle reveals more information than a traditional decryption or signature oracle, however, it does not possess more secret information. The reason being that, knowing one RSA private/public key pair is equivalent to knowing the factorization of n, which allows one to compute a private exponent for any given public exponent. A SEM in IB-mRSA can be viewed as a multi-key oracle in a sense that "bad" users can act as attackers and use the SEM to decrypt/sign any number of chosen messages. Although no formal security proof is offered in this paper, we were unable to find any attacks in this model. (Albeit, it seems that the security of the *multi-key oracle* is somehow tied to the *Strong RSA* Assumption [6].)

Key Generation In Section 3.1, we directly use the numerical value of an email address as a public key. The main requirement for an integer to form a proper RSA public exponent e is that it should be relatively prime with $\phi(n)$. Clearly, not all ASCII strings satisfy this requirement.

Since we choose $n = (2p' + 1)(2q' + 1)$, we have $\phi(n) = 4p'q'$. Both p' and q' are large primes (ca. 511 bits) while a typical email address is typically at most 320 bits, which is much smaller than p' and q'. In addition, note that all email addresses are converted into odd integers in *Algorithm IBE mRSA.Key*. Thus the derived e (which is odd) is not divisible by any of: $\{2, 4, p', q'\}$. In other words, e is relatively prime with $\phi(n)$.

Even if a very long email address is used, the probability of $gcd(e, \phi(n)) > 1$ is negligible. The reason is that all odd integers which are not relatively prime with $4p'q'$ are: $\{p', 3p', \ldots, (4q' - 1)p'\}$ and $\{q', 3q', \ldots, (4p' - 1)q'\}$. There are $2(p' + q' - 1)$ such integers. Thus, for a random odd integer $r < \phi(n)$:

$$Pr[gcd(r, \phi(n)) > 1] = \frac{(p' + q' - 1)}{2p'q'}$$

which is overwhelmingly small.

As pointed out in [3], a secure RSA setting requires that the decryption key to be large enough. Usually it should be at least larger than $n^0.5$, which is relatively the size of p and q. This is guaranteed by choosing short email address. For example, if e is at most 320 bits, the corresponding d should be longer than 512 bits, in case of a 1024-bit modulus.

Forward Security One of the motivations for research on ID-based cryptosystems is to provide forward security [1]. A usual forward security technique involves using a time-variant parameter along with user identifier, such that the derived key changes with time, e.g., every day or every month.

IB-mRSA can provide forward security in the same manner. Rather than computing e from just an email address, current date can be appended. In this case, the key distribution process in Section 3.1 should be revised. Instead of getting the signed and encrypted key bundle when joining an organization, the user sends a request to the CA when it received the first encrypted message in the current period. This causes the CA to generate and distribute the appropriate keys.

This is clearly an expensive process since the CA should be on-line and ready for users' requests. In addition, the CA needs to compute all non-revoked keys in every period. Thus, forward secure IB-mRSA is more suitable for organizations where IB-mRSA is lightly used and there exists an secure channel between users and the CA to protect key generation requests.

Organization Certificate and User Certificate Once the common RSA modulus n and the mapping function $f()$ is chosen, the CA issues and publishes a domain (attribute) certificate which includes these two values. In addition, the certificate also contains the organization/domain name (e.g. `@ics.uci.edu`).

User certificates are, as mentioned before, optional in IB-mRSA. They actually bridge IB-mRSA with plain RSA and mRSA. With the individual certificates, IB-mRSA becomes fully compatible with normal mRSA. The sender can get the public key from the individual certificate if it trusts it, or rely on the domain certificate and receiver's email address to compute the public key.

5 Implementation

To test our assertions and gain experimental and practical experience, we implemented the IB-mRSA scheme. Our implementation includes a IB-mRSA library as well as a fully functional email plug-ins for Eudora[12] and Microsoft Outlook[10]. It is freely available from the following web address:

`http://sconce.ics.uci.edu/SUCSES`

This implementation, incidentally, is part of the of the SUCSES project code release and borrows much of the code from the mRSA package. We re-used the basic mRSA functions, including user and SEM bundle generation as well as decryption and signature procedures.

Our Eudora and Outlook plug-ins allow the sender to encrypt outgoing email to all clients in the same domain using only one domain (organizational) certificate. When the user is ready to send, the plug-in reads the recipient's email address and looks up the organization certificate by using the domain name in the email address. A screen snapshot of the Eudora plug-in is shown in Figure 1.

When an email message encrypted with IB-mRSA is received, an icon for IB-mRSA is dislayed in the message window. To decrypt the message, the user just clicks on the

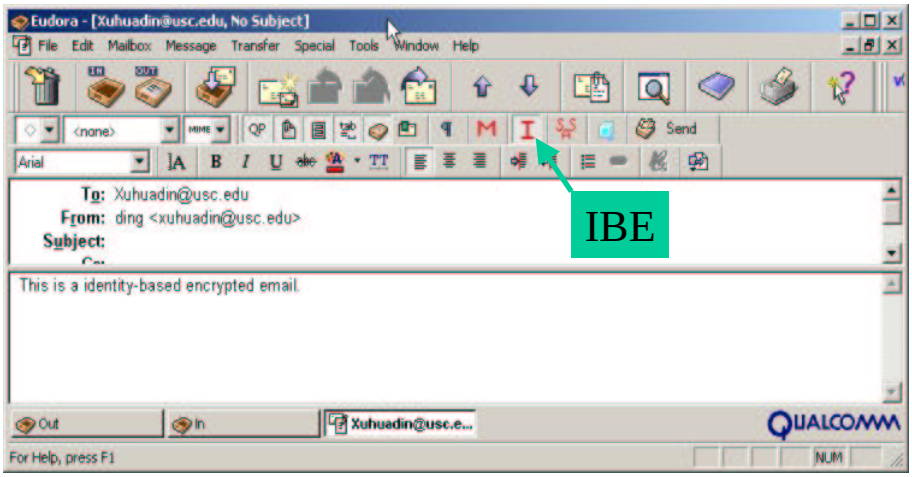

Fig. 1. Eudora IBE Plugin

IB-mRSA icon. The plug-in then contacts the user's SEM to get a partial decrypted message (provided the user is not revoked). This is basically the same process as plain mRSA [4].

6 Performance

When plain RSA is used for encryption, the public encryption exponent e is typically a small integer with only a few bits set to 1. One example is the popular OpenSSL toolkit [11] which uses $65,537$ as the default public key value for RSA certificates. Encryption with such small exponents can be accelerated with specialized algorithms for modular exponentiation. However, in identity-based systems, there is no such luxury of choosing special exponents. Therefore, e is a larger integer with likely higher number of 1 bits.

Keys	RSA Modulus 1Kb	RSA Modulus 2Kb	RSA Modulus 4Kb
$65,537$	2.3 ms	3.6 ms	11.7 ms
xhding@isi.edu	2.8 ms	9.5 ms	32.1 ms
Alice.Smith@ics.uci.edu	4.8 ms	15.5 ms	53.7 ms

Table 1. Performance Comparison of Different Encryption Keys

We ran some simple tests to measure the cost of IB-mRSA encryption for public keys derived from email addresses. The encryption was tested using Linux 2.4 version of OpenSSL on an 800MHz PIII workstation. In our tests, we used: 1) "default" encryption exponent $65,537$ and 2) two other exponents derived from different email addresses. For each key, we encrypted the same message one thousand times and obtained the average. The results are depicted in Table 1.

¿From the results in Table 1, we note that encryption with email address-derived keys does not introduce significant added overhead. Although the IB-mRSA encryption overhead increases as the RSA modulus size grows, it is somewaht negligible for 1024-bit modulus (which is currently the most common size).

The decryption cost for IB-mRSA is identical to mRSA. The performance of mRSA has been reported on by Boneh, et al. in [4]. For example, a 1024-bit mRSA decryption costs around 30ms on an 800 MHz PIII, as compared to 7.5ms for plain RSA on the same platform. We note that this is still much cheaper than 90ms that is needed for Boneh/Frnklin IBE decryption (for 1024 bits of security on the same hardware platform).

7 IB-mRSA versus Boneh/Franklin IBE

Recently, Boneh and Franklin developed a novel and elegant identity-based encryption system (IBE) based on Weil Pairing [5]. IBE represents a significant advance in cryptography since the problem that it solved has been open for many years. However, IBE does not provide revocation of users' security capabilities. This is natural since it aims to avoid the use of certificates in the course of public key encryption. On the other hand, revocation is often necessary and even imperative.

The only way to obtain revocation in IBE is to require fine-grained time-dependent public keys, e.g., public keys derived from identifiers combined with time- or date-stamps. This has an unfortunate consequence of having to periodically re-issue all private keys in the system. Moreover, these keys must be (again, periodically) securely distributed to individual users. Therefore, the Public Key Generator (PKG, in IBE's parlance) must be on-line much of the time, or, at least, very often. Consequently, we observe that, when IBE is used to provide fine-grained revocation, a PKG is, for all practical purposes, equivalent to a SEM in mRSA and IB-mRSA.

Our conclusion is that, in functional terms, IB-mRSA seems to offer security equivalent to IBE (when the latter provides fine-grained revocation). This is because compromise of a PKG in IBE results in a total system break. The same happens upon a SEM compromise in IB-mRSA. Moreover, IB-mRSA offers some practical advantages over IBE.

First, IB-mRSA is fully compatible with plain RSA if (optional) individual certificates are used. Thus, it offers a smooth and natural transition from ID-based to normal cryptography. Also, IB-mRSA (which takes roughly 4-5 times less efficient than plain RSA) offers significantly better performance than IBE.

8 Future Work and Summary

In this paper, we described a simple, secure and efficient IB-mRSA scheme. IB-mRSA combines the convenience of identity-based encryption (thus greatly reducing the need for public key certificates) with the functionality of mediated RSA which provides fine-grained revocation.

IB-mRSA allows the sender (encryptor) to skip the costly checking of individual public key certificates. The tight control over users' security capability is inherited from

mediated RSA mechanism since the decryption protocol for IB-mRSA is essentially the same as in mRSA. Furthermore, IB-mRSA can be easily be extended to provide forward security.

Several issues remain for future work. First, we have not provided a formal proof that RSA (with OAEP padding) is secure against adaptive chosen ciphertext attack if the decryption oracle offers replies for multiple private keys with the same modulus. We also need to investigate alternative mapping functions in order to speed up the encryption with derived public keys.

References

1. R. Anderson. Invited lecture at the acm conference on computer and communication security (ccs'97), 1997.
2. M. Bellare and P. Rogaway. Optimal asymmetric encryption — how to encrypt with RSA. In A. D. Santis, editor, *Advances in Cryptology – EUROCRYPT '94*, number 950 in Lecture Notes in Computer Science, pages 92–111. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 1995. Final (revised) version appeared November 19, 1995. Available from `http://www-cse.ucsd.edu/users/mihir/papers/oaep.html`.
3. D. Boneh. Twenty years of attacks on the RSA cryptosystem. *Notices of the American Mathematical Society (AMS)*, 46(2):203–213, 1999.
4. D. Boneh, X. Ding, G. Tsudik, and C. M. Wong. A method for fast revocation of public key certificates and security capabilities. In *10th USENIX Security Symposium*, Washington, D.C., Aug. 2001. USENIX.
5. D. Boneh and M. Franklin. Identity-based encryption from the Weil Pairing. In J. Kilian, editor, *Advances in Cryptology – CRYPTO '2001*, number 2139 in Lecture Notes in Computer Science, pages 213–229. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 2001.
6. E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In B. S. Kaliski, Jr., editor, *Advances in Cryptology – CRYPTO '97*, number 1294 in Lecture Notes in Computer Science. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 1997.
7. E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP is still alive! Record 2000/061, Cryptology ePrint Archive, Nov. 2000. Revised on Dec. 4, 2000. A revised version titled "RSA—OAEP is secure under the RSA Assumption".
8. R. Ganesan. Augmenting kerberos with pubic-key cryptography. In T. Mayfield, editor, *Symposium on Network and Distributed Systems Security*, San Diego, California, Feb. 1995. Internet Society.
9. P. Gemmel. An introduction to threshold cryptography. *RSA CryptoBytes*, 2(7), 1997.
10. Microsoft. Microsoft Outlook©, `http://www.microsoft.com`.
11. OpenSSL User Group. The OpenSSL Project Web Page, `http://www.openssl.org`.
12. Qualcomm. Qualcomm eudora mailer, `http://www.eudora.com`.

Experimenting with Server-Aided Signatures

Xuhua Ding, Daniele Mazzocchi and Gene Tsudik
Information and Computer Science Department
University of California, Irvine
{*xhding,dmazzocc,gts*}*@ics.uci.edu*

November 13, 2001

Abstract

This paper explores practical and conceptual implications of using Server-Aided Signatures (SAS). SAS is a signature method that relies on partially-trusted servers for generating public key signatures for regular users. Besides its two primary goals of 1) aiding small, resource-limited devices in computing heavyweight (normally expensive) digital signatures and 2) fast certificate revocation, SAS also offers signature causality and has some interesting features such as built-in attack detection for users and DoS resistance for servers.

1 Introduction

Digital signatures represent a basic building block for many secure applications. Their uses range from electronic commerce transactions to secure email, secure content (code, video, audio) distribution and other, more specialized applications such as document notarization. Traditionally, digital signatures are based on asymmetric (public key) cryptographic techniques which, at least in some settings, makes them computationally expensive.

While digital signatures are rapidly becoming ubiquitous, one of the major recent trends in computing has been towards so-called "smart" devices, such as PDAs, cell phones and palmtops. Although such devices come in many shapes and sizes and are used for a variety of purposes, they tend to have one feature in common: limited computational capabilities and equally limited power (as most operate on batteries). This makes them ill-suited for complex cryptographic computations such as large number arithmetic present in virtually all public key constructs.

Furthermore, in many envisaged setting, such as cell telephony and wireless web access, personal devices are in constant contact with a fixed, wired infrastructure. Consequently, access to more powerful (in terms of both CPU speed and not dependent on batteries) computing platforms is available to end-users.

At the same time, increased use of digital signatures accentuates the need for effective revocation methods. Revocation of cryptographic credentials and certificates has been an issue for a long time. However, only now the problem is becoming truly visible, e.g., the recent Verisign fiasco where a wrong certificate was issued (ostensibly to Microsoft) and its subsequent revocation was both slow and painful. Furthermore, current CRL-based revocation methods scale poorly and are not widely used in practice. For example, most current web browsers do not bother checking CRLs; only the upcoming Windows XP has some rudimentary CRL-checking facilities.

Effective revocation not only useful but vital in some organizational settings (e.g., government and military) where digital signatures are used on important electronic documents and in accessing critical resources. Consider a situation when a trusted user (Alice) does something that warrants immediate revocation of her security privileges. Alice might be fired, transferred or she may suspect that her private key has been compromised. Ideally – immediately following revocation – no one should be able to perform any cryptographic operations involving Alice's certificate, i.e., sign with her private key.

In addition, when a cryptographic certificate is revoked (or simply expires) digital signatures generated prior to revocation (or expiration) may need to remain valid. This is difficult to achieve with current revocation methods since CRLs (and similar methods like OCSP [1]) do not provide a secure means of distinguishing between pre- and post-revocation signature activity. The only way to do so is by using a secure timestamping service for all signatures. Although a secure timestamping service may provide a secure means of distinguishing between pre- and post-

revocation signature, it has not been widely adopted due to its prohibitive cost. Finally, we note that compromise of a private key can lead to an unlimited number of fraudulent signatures being generated and distributed by the adversary. As often happens in the event of compromise, contact with the revocation authority (CA) may not be immediate, e.g., in a sporadically connected wireless network. Therefore, it is important to find a way to limit potential damage.

In this paper we present a method, called Server-Aided Signatures (SAS), that is designed to addresses the aforementioned issues. Its goals are three-fold:

1. Assist small, limited-power devices in computing digital signatures
2. Provide fast revocation of signing capability
3. Limit damage from potential compromise

The rest of the paper is organized as follows. Next section provides a brief synopsis of our work and its contributions. Section 5 describes the SAS method in greater detail; it is followed by the security analysis in Section 6. Denial of service issues are addressed in Section 7. Then, implementation and performance measurements are discussed in Section 8. The paper concludes with the summary of benefits and drawbacks of SAS.

2 Synopsis

The signature method (SAS) discussed here is based largely on a weak non-repudiation technique due to Asokan et al. [2]. The most notable feature of the SAS method is its **on-line** nature. Specifically, each SAS signature is generated with the aid of a partially-trusted server called a SEM (short for **SE**curity **M**ediator). This feature can be viewed as a mixed blessing. Although it offers a number of benefits which are summarized below, the requirement for on-line help for each signature is clearly a burden. We discuss the drawbacks, both real and perceived, in Section 9.

Informally, a SAS signature is computed as follows (see also Figure 1):

- First, a prospective signer (Alice) contacts her SEM and provides the data to be signed as well as a one-time *ticket*.
- SEM checks Alice's revocation status and, if not revoked, computes a half-signature over the data as well as other parameters (including the one-time ticket). SEM then returns the results to Alice.
- Alice verifies SEM's half-signature and produces her own half-signature. Put together, the two respective half-signatures constitute a regular, full SAS signature. This signature is accompanied by SEM's and Alice's certificates.

The two half-signatures are inter-dependent and each is worthless in and of itself. This is despite the SEM's half-signature being a traditional digital signature: in the context of SAS, a traditional signature computed by a SEM is not, by itself, a SAS signature. The half-signature computed by a user (Alice, in our example) is actually a one-time signature [3].

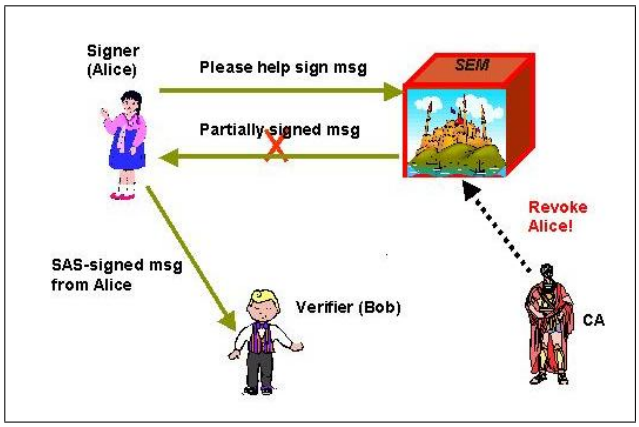

Figure 1: SAS architecture

Verifying a SAS signature is easy: verifier (Bob) obtains the signature and verifies the two halves along with the two accompanying certificates.

The main idea is that a SEM, albeit only partially trusted, is more secure, and much more capable (in terms of CPU and power consumption) than an average user. It can therefore serve a multitude of users. Also, because of its "superior" status, SEM is much less likely to be revoked or compromised. Since a signer (Alice) is assumed to have much less computing power then a SEM, the latter performs the bulk of the computation, whereas, Alice does comparatively little work. In the event that Alice's certificate is revoked, the SEM simply refuses to perform any further signatures on Alice's behalf. (See Figure 1.) Thus, revocation is both implicit and fast. However, this does not obviate the need for Certificate Revocation Lists (CRLs) since Alice's certificate may be revoked after some fraudulent signatures have been already generated. A CRL may still be necessary to convey to all verifiers the exact time of revocation and hence to sort out pre- and post-revocation signatures.

The general system model of SAS is a good fit for many mobile settings. For example, as mentioned in

Section 1, cell phones are only usable when in touch, via a nearby base station, with a fixed infrastructure. Each phone-call requires communication with the infrastructure. This communication can be overloaded to piggyback SAS protocol messages.

3 Related Work

The SAS method is based on a weak non-repudiation technique proposed by Asokan et al. in [2]. In very general terms, SAS is an instantiation of a mediated cryptographic service. Recent work by Boneh et al. [4] on mediated RSA (mRSA) is another example of mediated cryptography. mRSA provides fast revocation of both signing and decryption capability. However, the computation load on the client end is increased in mRSA, which is something that SAS aims to minimize.

In [5] Reiter and McKenzie propose a the same additive splitting technique to improve the security for portable devices where the private-key operations are password-protected. Recently, they also proposed another scheme for the more challenging problem of mediated (2-party) DSA signatures [6]. Ganesan[7] also exploited (earlier, in 1996) the same idea for improving Kerberos security as part of the Yaksha system.

Another way to look at SAS is as an instantiation of "hybrid" multi-signatures [8]. Viewed more broadly, the SAS method can be included in the more general framework of threshold cryptography[9] and secure multi-party computation[10].

There is also much related work on the topic of certificate revocation; including CRLs, Δ-CRLs, CRTs, 2-3 lists and skip-lists. This is reviewed in more detail in Appendix B.

4 Background

In this section we go over some preliminaries necessary for the remainder of the paper.

4.1 Hash Functions

Informally, a **one-way** function $f()$ is a function such that, given an input string x it is easy to compute $f(x)$, whereas, given a randomly chosen y, it is computationally infeasible to find an x such that $f(x) = y$. A one-way **hash** function $h()$ is a one-way function that operates on arbitrary-length inputs to produce a fixed length digest. If $y = h(x)$, y is commonly referred to as the *hash of* x and x is referred to as the *pre-image* of y. A one-way hash function $h()$ is said to be **collision-resistant** if it is computationally hard to find any two distinct input strings x, x' such that $h(x) = h(x')$.

Several secure and efficient collision-resistant one-way hash functions have been proposed, e.g., SHA or MD5 [11]. In the rest of the paper, $h()$ denotes a collision-resistant one-way hash function.

A collision-resistant one-way hash function can be recursively applied to an input string. The notation $h^i(x)$ is the result of applying $h()$ i times starting with the input x, that is:

$$\underbrace{h^i(x) = h(h(\ldots h(h(x))\ldots))}_{i \text{ times}}$$

Recursive application results in a *hash-chain* generated from the original input:

$$x = h^0(x), h^1(x), \ \ldots, \ h^n(x)$$

Hash chains have been widely used since early 1980-s starting with the well-known Lamport's method [12].

4.2 Model and Notation

We distinguish among 3 types of entities:

- *Regular Users* – entities who generate and verify SAS signatures.
- *Security Mediators (SEMs)* – partially-trusted entities assisting regular users in generating SAS signatures.
- *Certification Authorities (CAs)* – trusted off-line entities that issue certificates and link the identities of regular users with SEMs.

SEMs and CAs are verifiable third parties from the users' point of view.

All participants agree on a collision-resistant one-way hash function family $\mathcal{H}$ and a digital signature scheme. In SAS, the latter is fixed to be the RSA scheme [13]. Furthermore, each signer (Alice) selects a "personalized" hash function $h_A() \in \mathcal{H}$. In essence, $h_A()$ can be thought of as a keyed hash (e.g., [14]) with a known key set to the identity of the signer. When applied recursively, we also include the index of the hash function link in each computation, i.e., $h_A^i(x)$ can be thought of as a keyed hash where the known key is the concatenation of the signer's identity (Alice) and the index of the link, i.

In order to minimize computation overhead for regular users, $h()$ must be efficient and the digital signature scheme must be efficient for verifiers. (This is because, as will be seen below, verification is done

by regular users, whereas, signing is done by much more powerful SEMs.) SHA and MD5 are reasonable choices for the former, while RSA [13] satisfies the efficient verification requirement when used with a small exponent such as 3, 17 or 65,537.

4.3 Communication Channel

We assume that the communication channel between each user and a SEM is **reliable** (but neither private nor authentic). Reliability of the channel implies that the underlying communication system provides sufficient error handling to detect, with overwhelming probability, all corrupted packets. One way to achieve this is by having each protocol packet accompanied by its hash. Furthermore, timeouts and retransmissions are likewise handled by the communication system with the assumption that a packet eventually gets through.

We note that, even if the user is disconnected from the network[1] after sending a signature request to its SEM and before receiving a reply, the user will eventually obtain the correct reply (if the request ever reached the SEM) whenever the communication channel is re-established. Specifically, as described in the next section, a SEM always replies with the last signature it computed for a given user.

5 SAS Description

We now turn to the detailed protocol description.

5.1 Setup

To become a SAS signer, Alice first generates a secret quantity SK_A^0 randomly chosen from the range of $h_A()$. Starting with this value, Alice computes a hash-chain:

$$\{\ SK_A^0, SK_A^1, \ldots SK_A^{n-1}, SK_A^n\ \} \text{ where}$$

$$SK_A^j = h_A^j(SK_A^0) = h_A(SK_A^{j-1}) \text{ for } 1 \leq\ j \leq\ n$$

The last value, SK_A^n, is referred to as Alice's *SAS root key*. It subsequently enables Alice to produce $(n-1)$ SAS signatures.

Each SEM is assumed to have a secret/public RSA key-pair (SK_{sem}, PK_{sem}) of sufficient length. (We use the notation $[x]^{sem}$ to denote SEM's signature on string x). Each CA also has its own key-pair much like any traditional CA. In addition to its usual role of issuing and revoking certificates a CA also maintains a mapping between users and SEMs that serve them. This relationship is many to one, i.e., a SEM serves a multitude of users. Exactly how many depends on many factors, such as: SEM's hardware platform, average user signature request frequency, network characteristics, etc. We expect the number and placement of SEMs in an organizational network to closely resemble that of OCSP Validation Agents (VAs) [1].

In order to obtain a SAS certificate $Cert_A$, Alice composes a certificate request and submits it to the CA via some (usually off-line) channel. Alice's SAS certificate has, for the most part, the same format as any other public key certificate; it includes values such as the holder's distinguished name, organizational data, expiration/validity dates, serial number, public token key, and so forth. Additionally, a SAS certificate contains two other fields:

1. Maximum number of signatures n that the enclosed public key can be used to generate, and
2. Distinguished name and certificate serial number of the SEM serving the certificate holder.

Once issued, Alice's SAS certificate $Cert_A$ can be made publicly available via a directory service such as LDAP [15].

5.2 SAS Signature Protocol

The protocol proceeds as follows. (In the initial protocol run the signature counter $i = n-1$; it is decremented after each run. This counter is maintained by both SEM and Alice.)

Step 1. Alice starts by sending a request containing: $[\text{Alice}, m, i, SK_A^i]$ to its assigned SEM. If Alice does not wish to reveal the message to the SEM, m can be replaced with a suitable keyed (or, more accurately, randomized) hash such as the well-known HMAC [14]. (In that case, Alice would send $HMAC_r(m)$ where r is a one-time random value used a key in the HMAC computation.)

Alice may also (optionally) enclose her SAS certificate.

Step 2. Having received Alice's request, SEM obtains $Cert_A$ (either from the request or from local storage) and checks its status. If revoked, SEM replies with an error message and halts the protocol. Otherwise, SEM compares the signature index in the request to its own signature counter. In case of a mismatch, SEM replies to Alice with the

[1]This can happen if a wireless device, e.g., a cell phone, is momentarily out of range of any base station.

lowest-numbered half-signature produced in the last protocol run and aborts.

Next, SEM proceeds to verify the received public key (SK_A^i) based on Alice's SAS root key contained in the certificate. (If this is Alice's initial request, the signature counter is initialized to Having received Alice's request, SEM obtains $Cert_A$ (either from the request or from local storage) and checks its status. If revoked, SEM replies with an error message and halts the protocol. Otherwise, SEM compares the signature index in the request to its own signature counter. In case of a mismatch, SEM replies to Alice with the lowest-numbered half-signature produced in the last protocol run and aborts. Specifically, SEM checks that $h_A^{n-i}(SK_A^i) = SK_A^n$. In case of a mismatch, SEM replies to Alice with the last recorded half-signature and aborts the protocol.

Next, SEM signs the requested message with its private key to produce: $[Cert_A, m, i, SK_A^i]^{SEM}$. Other attributes may also be included in SEM's half-signature, e.g., a timestamp. SEM decrements Alice's signature counter, records the half-signature and returns the latter to Alice.

In the above, SEM assures that – for a given SAS certificate – exactly one signature is created for each $[i, SK_A^i]$ tuple. We refer to this property as the **SAS Invariant**.

Step 3. Alice (who is assumed to be in possession of SEM's certificate at all times) verifies SEM's half-signature, records it and decrements her signature counter. If SEM's half-signature fails verification or its attributes are wrong (e.g., it signs a different message than m or includes an incorrect signature counter $j \neq i$), Alice aborts the protocol and concludes that a hostile attack has occurred.[2] (See Section 7 below.)

Finally, Alice's SAS signature on message m has the following format:

$$SIG_i = [Cert_A, m, i, SK_A^i]^{SEM}, SK_A^{i-1}$$

The second part, namely SK_A^{i-1}, is Alice's half-signature. As mentioned earlier, it is actually a one-time signature: $h_A(SK_A^{i-1}) = SK_A^i$.

Note that Alice must use her one-time keys in strict sequence. In particular, Alice must not request a SEM half-signature using SK_A^{i-1} unless, in the last protocol run, she obtained SEM's half-signature containing SK_A^i.

[2]Our communication channel assumption rules out non-malicious packets errors.

5.3 SAS Signature Verification

SAS signature verification comes in two *flavors*: *light* and *full*. The particular choice depends on the verifier's trust model. Recall that the philosophy of SAS is based on much greater (yet not unconditional) trust placed in a SEM than in a regular user. If a verifier (Bob) fully subscribes to this, i.e., trusts a SEM more than Alice, he can chose light verification. Otherwise, if Bob is equally suspicious of SEMs as of ordinary users, he can choose full verification.

Light verification involves the following steps:

1. Obtain and verify[3] $Cert_{SEM}$
2. Verify SEM's RSA half-signature: $[Cert_A, m, i, SK_A^i]^{SEM}$
3. Verify Alice's half-signature: $h_A(SK_A^{i-1}) \stackrel{?}{=} SK_A^i$

Full verification requires, in addition:

4. Verify $Cert_A$
5. Check that $i < n$
6. Verify Alice's SAS root key: $h_A^{n-i}(SK_A^i) \stackrel{?}{=} SK_A^n$

Note that light verification does not involve checking Alice's SAS certificate. Although this may seem counter-intuitive, we claim that SAS signature format (actually SEM's half-signature) already includes $Cert_A$ as a signed attribute. Therefore, for a verifier who trusts the SEM, step 2 above implicitly verifies $Cert_A$.

It is easy to see that, owing to the trusted nature of a SEM and the **SAS Invariant**, light verification is usually sufficient. However, if a stronger property (such as non-repudiation) is desired, full verification may be used.

5.4 State and Registration

As follows from the protocol description above, both Alice and the SEM maintain state. Alice's SAS state amounts to the following:

$$Cert_A, Cert_{SEM}, SK_A^0, i, \{SIG_n, ..., SIG_{n-i-1}\}$$

[3]This may be done infrequently.

The first three values are self-explanatory. The fourth is Alice's current signature counter, (i), and the rest is the list of previously generated signatures for the same $Cert_A$. The state kept by the SEM (for each user) is similar:

$$Cert_A, i, \{SIG_n, ..., SIG_{n-i-1}\}$$

The amount of state might seem excessive at first, especially considering that some users might be on small limited-storage devices. There are some optimizations, however. First, we note that Alice can periodically off-load her prior signatures to some other storage (e.g., to a workstation or a PC when the PDA is charging). Also, it is possible to drastically reduce state maintenance for both users and SEMs if successive signatures are accumulated. For example, each SEM's half-signature can additionally contain the hash of the last prior SAS signature. This optimization results in storage requirements comparable to those of a traditional signature scheme.

Registration in SAS can be done either off- or on-line. In the off-line case, SEM obtains Alice's SAS certificate via manual (local or remote) installation by an administrator or by fetching it from the directory service. To register on-line, Alice simply includes her SAS certificate as an optional field in the initial SAS signature request to the SEM. Before processing the request as described above, the SEM checks if the same certificate is already stored. If not, it installs in the certificate database and creates a new user entry. (See Figure 2.)

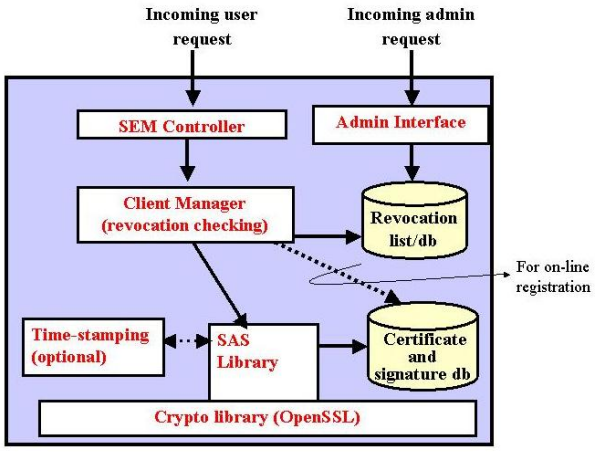

Figure 2: SEM architecture

6 Analysis

We now consider the efficiency and security aspects of the SAS signature method.

6.1 Efficiency

The cost of our signature protocol can be broken up as follows:

1. Network overhead: round-trip delay between Alice and SEM
2. SEM computation: signature computation plus other overhead (including hash verification of user's one-time public key, database processing, etc.)
3. User computation: verification of the SEM half-signature and other (commitment to storage) overhead.

Clearly, (1) and (3) are extra steps as compared with a traditional signature method. The extra cost of light signature verification (referring to the steps in the previous section) is only in Step 3 which consists of a single hash operation. Full verification costs an additional certificate validation (Step 4) as well as $(n - i)$ hash operations in Step 5.

6.2 Security Analysis

We claim that the SAS signature method achieves the same security level as a traditional digital signature scheme if SAS signature and verification protocols are executed correctly. Due to space limitations, we only present an informal security analysis.

To forge a SAS signature, an adversary can attempt to:

TYPE 1: forge a SEM's half-signature (i.e., an RSA signature) or

TYPE 2: find a quantity SK_A^* such that $H(SK_A^*) = SK_A^i$. Recall that SK_A^i is included in SEM's half-signature.

Clearly, a TYPE 1 attack is an attack on the underlying signature scheme, i.e., RSA, and, as such, is not specific to the SAS method. Therefore, we only consider TYPE 2 attacks. However, finding SK_A^* implies a successful attack on either the collision-resistance or the one-wayness property of the underlying hash function $h_A()$. Even we were to allow the possibility of the adversary mounting a successful TYPE 2 attack, the scheme remains secure if *full*

verification is used. (Recall that full verification includes not only checking $H(SK_A^*) \stackrel{?}{=} SK_A^i$ but also $h_A^{n-i}(SK_A^i) \stackrel{?}{=} SK_A^n$.)

We observe that, in any practical digital signature scheme, a collision-resistant one-way hash function is first applied to the message in order to produce a fixed-length digest which is then signed. Hence, a successful TYPE 2 attack on a SAS signature is, at the same time, an attack on the digital signature scheme.

6.3 Disputes

In case of a dispute between a signer (Alice) and a verifier (Bob), the latter submits the disputed SAS signature to an unbiased arbitrator who starts by verifying the following:

- Alice's and SEM's certificates are valid and certified by a CA.
- SEM's half-signature is valid.
- Alice's one-time key is a hash pre-image of the value in SEM's half-signature.
- The SAS root key in $Cert_A$ can be derived from the one-time public key by repeated hashing.

This is essentially the full SAS signature verification as described earlier. If any of the above steps fails, the arbitrator rules in Alice's favor. Otherwise, Bob wins the dispute.

Assuming the above procedure succeeds, Alice is asked to produce a different SAS signature with the same one-time key (i.e., same one-time signature). If Alice can come up with such a signature (meaning that the message signed is different from the one in the disputed signature), the arbitrator concludes that Alice's SEM cheated or was compromised. This conclusion is based on the apparent violation of the **SAS Invariant**. If Alice fails to produce a different signature, the arbitrator concludes that Alice attempted to cheat.

7 Denial of Service

The SAS signature protocol, unlike traditional signature schemes, involves multiple parties and communication. It is therefore subject to Denial of Service (DoS) attacks. Since we assume that the communication channel is reliable (cf. Section 4.3), only hostile DoS attacks are of interest. Also, our channel assumption states that all messages eventually get through; thus, attacks on the communication media are ruled out.

There are two types of DoS attacks: user attacks and SEM attacks. The purpose of a user attack is to deny service to a particular user whereas the purpose of a SEM attack is to deny service to all users served by a SEM. User attacks can be further divided into request and reply attacks. Request attacks involves modifying (or injecting) a user's signature request and a reply attack – modifying a SEM's reply.

7.1 User Attacks

Suppose that an adversary (Eve) intercepts the signature request and mounts a request attack. In this case, SEM receives a request that is perfectly legitimate (well-formed) from its point of view. It proceeds to sign it and send the signed reply back to Alice. Clearly, Alice discards the reply because it contains a signature for a different message. If Eve prevents the reply from reaching Alice, she gains no advantage since, as explained above, forging a signature requires Eve to come up with a one-time public key which she cannot do without breaking the hash function. Even if the reply does not arrive immediately, according to our communication assumption, it eventually reaches Alice who promptly detects an attack.

A slight variation on the above occurs when Eve has in her possession the last SAS signature generated by Alice. In this case, Eve can contact Alice's SEM with a well-formed request and without Alice's knowledge, i.e., Alice is off-line. However, this attack results in the same outcome as the above. This is because, eventually, Alice requests a new signature and SEM replies with the last (signed) reply. Alice, once again, detects an attack.

We note that these attacks can be prevented: one way to do so is for Alice not to reveal her i-th signature until $(i-1)$-st signature is computed. In other words, every other signature would be used strictly for this purpose. Then, if we suppose that Alice-SEM communication is private, revealing SIG_i to Bob (or Eve) is safe since a successful request to Alice's SEM would require knowledge of SK_{i-1} which Alice does not reveal until the next signature is requested. Yet another solution is to use a second, different hash chain for the sole purpose to authenticate Alice's requests to the SEM.

All in all, request attacks, while possible, are detected by the SAS signature protocol due to its "fail-stop" property: any manipulation of the signature request is detected by the user who can then invalidate its own certificate.

User reply attacks are comparatively less effective. If Eve modifies SEM's reply, short of forging an RSA signature, Alice detects that the reply is not what she expected and continues re-transmitting her signature request.

7.2 SEM Attacks

By virtue of serving a multitude of regular users, a SEM is a natural DoS attack target. This is not unique to SAS. For instance, it is easy to mount an effective DoS attack against an OCSP [1] (or even worse, a TSP [16]) server. It suffices for the adversary to flood the victim server with well-formed requests, i.e., requests for which the server is "authoritative" in OCSP. Since the server must digitally sign all replies, it will slowly grind to a halt.

In SAS, it is appreciably more difficult for the adversary to launch this type of an attack. The stateful nature of the SEM requires each signature request to be well-formed: it must contain the expected value of the current one-time public-key, i.e., the pre-image of the previously used public-key. All other requests are promptly discarded.

Therefore, in order to force the SEM to perform any heavy-weight tasks (of which signing is really the only one), the adversary must mount simultaneous user request attacks on as many users as possible thus hoping to flood the SEM. However, even if this were possible, the attack would quickly subside since the SEM will only perform a single signature operation per user before demanding to see a pre-image (next one-time public key). As we already established, finding the pre-image of the last signed one-time public key is computationally infeasible.

7.3 Loss of State

As SAS requires a non-trivial amount of state to be maintained by both users and SEMs, we need to consider the potential disaster scenarios that result in a loss of state.

Suppose that Alice looses all records of her prior signatures along with the signature counter. We further assume that she still has possession of her SAS certificate and the secret hash chain seed. Since these two values are fairly long-term, it is reasonable for Alice to store them in more permanent storage. Because of the "amnesia", Alice will attempt to obtain the initial signature from the SEM. Since SEM has retained all relevant state, it will reply with the last half-signature (including SEM's signature counter) generated for Alice's SAS certificate. Once she verifies the reply, Alice will realize her loss of state and resort to off-line means. However, if a malicious SEM is aware of Alice's loss of state, it can use this to its advantage by forging with impunity Alice's signatures.

If Alice looses her entire storage, including the SAS certificate, the consequences are not particularly dire. The SEM will simply keep state of Alice's "orphan" certificate until it eventually expires.

Any loss of SEM's state is much more serious. Most importantly, if the SEM looses all state pertaining to Alice's SAS certificate, the **SAS Invariant** property can no longer be guaranteed. (Consider, for example, malicious Alice re-establishing state of her SAS certificate on the SEM and then obtaining n signatures with the same hash chain.)

7.4 SEM Compromise

SEM compromise is clearly the greatest risk in SAS. The adversary who gains control of a SEM can un-revoke or refuse to revoke SAS user certificates. Moreover, it becomes possible to produce fraudulent user signatures: since state is kept of all prior SAS signatures (corresponding to active SAS certificates), the adversary can sign on behalf of Alice for each (SK_A^i, SK_A^{i-1}) pair found in SEM's storage.

Nonetheless, a defrauded SEM user can still have recourse if she faithfully keeps state of all prior SAS signatures. Referring to the SAS dispute resolution procedure, when an arbitrator is presented with two distinct and verifiable SAS signatures for the same (SK_A^i, SK_A^{i-1}) pair, he concludes that the SEM has attempted to cheat.

7.5 Suicide in SAS

In order to provide rapid and effective response to potential attacks, SAS includes a way for the user to "self-revoke" a SAS certificate. This is easily obtained by placing a new value (X.509 extension) in the SAS certificate. This value, referred to as the "suicide hash", is the hash of a randomly selected secret quantity generated by Alice when composing her certificate request. To self-revoke the certificate, Alice simply communicates the corresponding suicide pre-image to the SEM and the CA. As a result, the former simply stops honoring any further signature requests (pertaining to Alice's certificate) while the latter places a reference to the said certificate on the next CRL.

A similar technique has been suggested (with the value revealed by the CA instead) by Micali [17] as part of a proposal for an efficient revocation scheme.

8 Implementation and Experiments

To better understand the implications of using SAS and to obtain valuable experimental and practical data, we implemented the SAS scheme, first as a limping proof-of-concept prototype and, later, as a fully functional and publicly available package.

The implementation, for the most part, follows the protocol as presented in Section 5. The SAS certificate issuance is done strictly off-line: all users obtain their SAS certificates from the CA as described in Section 5.1. The newly issued certificates are either transferred to SEM off-line or piggybacked onto each user's initial SAS signature request. We limit our implementation discussion owing to space limitations; further details, including the SAS signature and SAS certificate formats can be found in Appendix A.

8.1 SAS Application Example: Eudora Plug-in

To demonstrate the ease and utility of the SAS signatures, we developed a plug-in (on top of the SAS user library [18]) for the popular Eudora [19] mailer.

When composing email, the sender simply clicks on the plug-in button. When ready to send, the plug-in reads the user's SAS certificate and extracts the SEM's address. It then communicates with the SEM to obtain a SAS signature on the email message. The resulting signed email is verified automatically by the Eudora plug-in on the receiver's side. Even if the receiver does not use Eudora, the SAS-signed email can be verified by any S/MIME capable email client such as Netscape Messenger or Microsoft Outlook. The verification, however, requires the receiver (verifier) to install a stand-alone SAS email verifier program. This program is registered as the viewer for the new MIME type (``x.SAS-signature'').

Figure 3 shows a screen snapshot of the Eudora message composition window when the user is ready to send a signed email. It is essentially the same as the normal Eudora screen except for the small SAS button at the toolbar along the top of the window. Figure 4 depicts a screen snapshot of the Eudora mailer showing a SAS-signed email message being received. The user is presented with a signature icon on the message screen; clicking on it causes the mailer to invoke the plug-in's verification function the output of which is displayed in the Figure 5.

To conserve space we omit the depiction of a user trying to sign email with a revoked certificate. In this case, the plug-in displays an error message informing

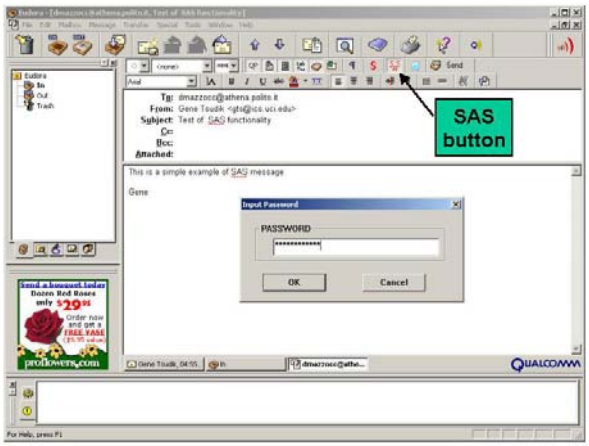

Figure 3: Snapshot of signer plug-in

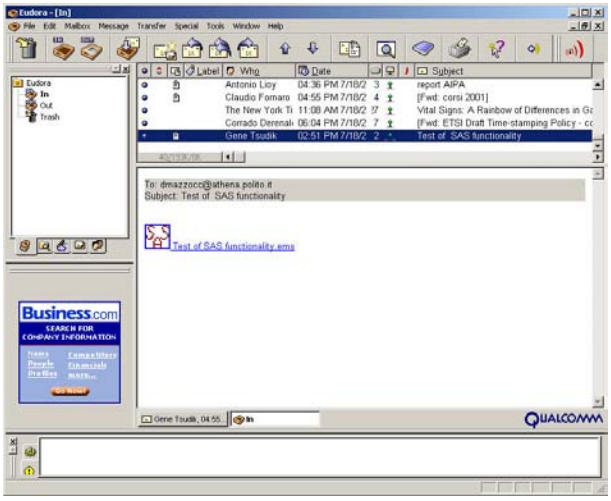

Figure 4: Verifier plug-in: signed email

the user of his certificate's demise. Further details on the Eudora plug-in can be found in Appendix A.

8.2 Experimental Results

As emphasized in the introduction, one of the main goals of SAS is to off-load the bulk of signature computation from the weak user to the powerful SEM. To validate the goals and experiment with the SAS implementation, we ran a number of tests with various hardware platforms and different RSA key sizes.

All experiments were conducted over a 100 Mbit Ethernet LAN in a lab setting with little, if any, extraneous network traffic. All test machines ran Linux version 2.2 with all non-essential services turned off. The hardware platforms ranged from a measly 233-MHz PI (Pentium I) to a respectable 1.2-GHz PIV

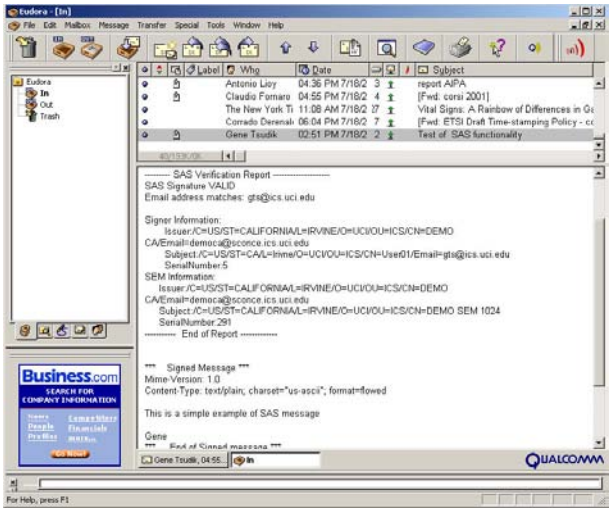

Figure 5: Verifier plug-in: verification

(Pentium IV). Note that we selected the lowest-end platform conservatively: only very high-end PDAs and palmtops approach 200-MHz processor speed; most are in the sub-100Mhz range. Our choice of the SEM platform is similarly conservative: a 933-MHz PIII. (At the time of this writing, 1.7-GHz platforms are available and affordable.)

Processor	**Key length (bits)**			
	1024	*2048*	*4096*	*8192*
PI-233 MHz	40.3	252.7	1741.7	12,490.0
PIII-500 MHz	14.6	85.6	562.8	3,873.3
PIII-700 MHz	9.2	55.7	377.8	2,617.5
PIII-933 MHz	7.3	43.9	294.7	2,052.0
PIV-1.2 GHz	9.3	58.7	401.2	2,835.0

Table 1: Plain RSA signature timings (ms)

First, we present in Table 8.2 plain RSA timings conducted with OpenSSL on the five hardware platforms. Table 8.2 illustrates the SAS timing measurements on the four user platforms with the SEM daemon running on a 933-MHz PIII. All SAS timings include network transmission time as well as SEM and user processing times. Finally, Table 8.2 shows the LAN round-trip communication delay between the user and the SEM, for different key sizes. The size of the signature request is determined by the digest size of the hash function, whereas, SEM's replies vary from roughly 164 bytes for 1024-bit RSA key to around $1,060$ bytes for an 8K-bit RSA key.

We purposely used fairly conservative platforms for both the SEM and test users. The slowest user platform is a 233-MHz Pentium I laptop which is significantly faster than a typical PDA or a cell phone. The motivation was to show that, even a relatively fast user CPU, the speedup from SAS is appreciable. Clearly, a more realistic scenario would involve, for example, a 60- to 100-MhZ PDA as the user platform and a 1.7- to 2-GhZ PIV as a SEM.

As is evident from Table 8.2, all four user platforms experience noticeable speed-up as a result of using SAS, as compared with plain RSA. It is not surprising that the two low-end clients (233-MHz and 500-MHz) obtain a factor 4 to 6 speed-up depending on the key size. It is interesting, however, that the seemingly most powerful client platform (1.2-GHz PIV) also experiences a small speed-up. However, looking at Table 8.2, it becomes clear that the 1.2-GHz PIV is not the fastest platform after all. The explanation for this oddity rests with the chip maker.

Processor	**Key length (bits)**			
	1024	*2048*	*4096*	*8192*
PI-233 MHz	13.3	52.4	322.5	2,143.4
PIII-500 MHz	9.1	46.3	302.0	2,070.2
PIII-700 MHz	8.5	45.1	299.0	2,059.6
PIV-1.2 GHz	8.5	45.4	299.0	2,061.0

Table 2: SAS signature timings (ms)

To summarize, as Tables 8.2 and 8.2 illustrate, despite large variances in the four clients' CPU speeds, the difference in SAS sign time is very small. Moreover, the SAS sign time is only slightly higher than the corresponding value for the SEM (PIII-933 MHz) in Table 8.2, meaning that – communication delay aside – a SAS client can sign almost as fast as the SEM. The reason is that, to obtain a SAS signature, a user's cryptographic computation (which dominates the overall time) amounts to message hashing and signature verification. Hashing is almost negligible as compared to public key operations. RSA signature verification is also quite cheap in comparison to signing since we use small public exponents.

Processor	**Key length (bits)**			
	1024	*2048*	*4096*	*8192*
PI-233 MHz	0.6	0.7	1.1	1.7
PIII-500 MHz	0.4	0.5	0.8	1.2
PIII-700 MHz	0.1	0.2	0.2	0.3
PIV-1.2 GHz	0.4	0.5	0.8	1.2

Table 3: Network round-trip delay (ms)

9 Benefits and Drawbacks

In summary, the SAS signature scheme offers several important benefits as described below: **Efficient Signatures.** As follows from the protocol description and our experimental results, the SAS signature scheme significantly speeds up signature computation for slow, resource-limited devices. Even where speed-up is not as clearly evident (e.g., with small key sizes), SAS signatures conserve CPU resources and, consequently, power, for battery-operated devices.

Fast revocation. To revoke a SAS certificate, it is sufficient for the CA to communicate to the correct SEM. This can be achieved, for example, with CA simply issuing a new CRL and sending it to the SEM. Thereafter, the SEM will no longer accept SAS signature requests for the revoked certificate.

We remark that, with traditional signature schemes, the user who suspects that his key has been compromised can ask the CA to revoke the certificate binding this key to the user. However, the adversary can continue *ad infinitum* to use the compromised key and the verification burden is placed on all potential verifiers who must have access to the latest CRL. With SAS, once the SEM is notified of a certificate's revocation, the adversary is no longer able to interact with the SEM to obtain signatures. Hence, potential compromise damage is severely reduced.

More secure signatures. Since only a SEM performs real RSA public key operations (key generation, signature computation), it can do so with stronger RSA keys than would otherwise be used by the users. Indeed, a small PDA-like device is much less likely to generate high-quality (or sufficiently long) RSA factors (p, q) and key-pairs than a much more powerful and sophisticated SEM.

Signature Causality. Total order can be imposed over all SAS signatures produced by a given user. This is a direct consequence of the hash chain construction and the **SAS Invariant**. In other words, total ordering can be performed using the monotonically increasing signature counter included in each SAS signature.

Dispute Resolution. Signature Causality can be used to provide unambiguous dispute resolution in case of private key compromise. Recall that the compromise of a private key in a traditional signature scheme results in chaos. In particular, all prior signatures become worthless unless the use of a secure timestamping service is explicitly mandated for all signer and signatures. In SAS, once the time of compromise is established, signatures can be easily sorted into pre- and post-revocation piles.

Attack Detection. As discussed in Section 7, an adversary can succeed in obtaining a single fraudulent half-signature (not a full SAS signature) by substituting a message of its own choosing in the user's signature request. This essentially closes the door for the adversary since it is unable to obtain further service (short of inverting the hash function). The real user will detect that an attacks has taken place the next time when it tries to run the SAS signature protocol with its SEM.

Limited Damage. Even if the entire SAS hash chain is compromised (i.e., an adversary obtains the seed of the hash chain), the damage is contained since the adversary can generate at most n signatures. Furthermore, a user whose hash chain is compromised will detect the compromise the very next time she attempts to contact the SEM. (This is because the SEM will reply with its last half-signature ostensibly computed for the requesting user.)

Alas, the SAS scheme has some notable drawbacks as well:

• Each SEM is a single point of failure and a performance bottleneck for the users it serves.

• As discussed in Section 7, a SEM signs (with RSA, to produce its half-signature) a response to every well-formed signature request. This feature can be exploited by an adversary in order to mount a DoS attack. However, even the best attack can succeed in making a SEM sign at most once for each user it serves. Of course, an adversary can still flood any SEM with malformed requests which can certainly render a SEM unavailable to legitimate users.

• Unlike other mediated or multi-party signature methods (such as mRSA or 2-party DSA), SAS signatures are not compatible with any other basic signature type. In other words, SAS signatures are not transparent to verifiers. Therefore, all potential verifiers must avail themselves of at least the SAS verification method.

• It is possible, but neither easy nor elegant, for

a user to switch among different SEMs in SAS. One way is to have multiple SAS certificates; one for a distinct SEM. Another way is to use on-line hand-over of a SAS certificate among two SEMs. Neither solution is particularly attractive due to the difficulty of replication of a statfule server. (In mRSA [4], for example, a user can swich among SEMs transparently, where SEM is stateless.)

• SAS involves on-going state retention for regular users and SEMs. This burden is particularly heavy for SEMs (users can off-load their state periodically) since they must keep complete signature histories for all users served.

Acknowledgements

We thank Dan Boneh for some useful discussions, Ignacio Solis for early prototyping of the SAS library and Yongdae Kim for comments on the draft of this paper. We also gratefully acknowledge the anonymous referees whose comments served to greatly improve the final version of this paper.

References

[1] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams, "RFC2560: Internet public key infrastructure online certicate status protocol - OCSP," June 1999.

[2] N. Asokan, G. Tsudik, and M. Waidner, "Server-supported signatures," *Journal of Computer Security*, vol. 5, no. 1, 1997.

[3] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Advances in Cryptology – CRYPTO '87* (C. Pomerance, ed.), no. 293 in Lecture Notes in Computer Science, (Santa Barbara, CA, USA), pp. 369–378, Springer-Verlag, Berlin Germany, Aug. 1988.

[4] D. Boneh, X. Ding, G. Tsudik, and B. Wong, "Instanteneous revocation of security capabilities," in *Proceeding of USENIX Security Symposium 2001*, Aug. 2001.

[5] P. MacKenzie and M. K. Reiter, "Networked cryptographic devices resilient to capture," in *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pp. 12–25, May 2001.

[6] P. MacKenzie and M. K. Reiter, "Two-party generation of dsa signatures," in *Advances in Cryptology – CRYPTO '01* (J. Kilian, ed.), no. 2139 in Lecture Notes in Computer Science, pp. 137–154, Springer-Verlag, Berlin Germany, Aug. 2001.

[7] R. Ganesan, "Argumenting kerberose with pubic-key crytography," in *Symposium on Network and Distributed Systems Security* (T. Mayfield, ed.), (San Diego, California), Internet Society, Feb. 1995.

[8] C. Boyd, "Digital mulitsignatures," *Cryptography and Coding*, pp. 241–246, 1989.

[9] Y. Desmedt and Y. Frankel, "Threshold cryptosystems," in *Advances in Cryptology – CRYPTO '89* (G. Brassard, ed.), no. 435 in Lecture Notes in Computer Science, (Santa Barbara, CA, USA), pp. 307–315, Springer-Verlag, Berlin Germany, Aug. 1990.

[10] O. Goldreich, "Secure multi-party computation (working draft)," 1998.

[11] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc, 1996.

[12] L. Lamport, "Password authentication with insecure communication," *Communications of the ACM*, vol. 24, pp. 770–772, Nov. 1981.

[13] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Journal of the ACM*, vol. 21, pp. 120–126, Feb. 1978.

[14] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *Advances in Cryptology – CRYPTO '96* (N.Koblitz, ed.), no. 1109 in Lecture Notes in Computer Science, pp. 1–15, Springer-Verlag, Berlin Germany, 1996.

[15] S. Boeyen, T. Howes, and P. Richard, "RFC 2559: Internet x.509 public key infrastructure operational protocols - LDAPv2," 1999.

[16] C. Adams, P. Cain, D. Pinkas, and R. Zuccherato, "Internet x.509 public key infrastructure time stamp protocol (tsp), draft-ietf-pkix-time-stamp-15.txt," May 2001.

[17] S. Micali, "Enhanced certificate revocation system," Tech. Rep. TM-542b, MIT/LCS, May 1996.

[18] "SAS plug-in web page," available at: *http://sconce.ics.uci.edu/sucses/*.

[19] "Qualcomm eudora mailer," available at: *http://www.eudora.com*.

[20] R. Laboratories, "Cryptographic message syntax standard," Public Key Cryptography Standards 7, RSA Laboratories, Redwood City, CA, USA, 1993. Available at URL: ftp://ftp.rsa.com/pub/pkcs/. 1993.

[21] R. Housley, W. Ford, W. Polk, and D. Solo, "RFC 2459: Internet x.509 public key infrastructure certificate and crl profile," Jan. 1999.

[22] "The openssl project web page," *http://www.openssl.org*.

[23] P. Kocher, "On certificate revocation and validation," in *Financial Cryptography – FC '98, Lecture Notes in Computer Science, Springer-Verlag, Vol. 1465*, pp. 172–177, 1998.

[24] M. Naor and K. Nissim, "Certificate revocation and certificate update," in *Proceedings 7th USENIX Security Symposium (San Antonio, Texas)*, Jan 1998.

[25] M. Goodrich, R. Tamassia, and A. Schwerin, "Implementation of an authenticated dictionary with skip lists and commutative hashing," in *Proceedings of DARPA DISCEX II*, 2001.

[26] W. Aiello, S. Lodha, and R. Ostrovsky, "Fast digital identity revocation," in *Advances in Cryptology – CRYPTO '98* (H. Krawczyk, ed.), no. 1462 in Lecture Notes in Computer Science, Springer-Verlag, Berlin Germany, Aug. 1998.

Appendix A: SAS Implementation Details

A.1 SAS Signature Format

The well-known PKCS#7 [20] standard defines a general cryptographic message syntax for digital signatures. In it, `SignerInfor` includes an optional set of signed attributes as well as an optional set of unsigned attributes. This flexibility allows us to easily extend the PKCS#7 signature syntax to accommodate SAS signatures. This is because a SAS signature can be viewed as a regular public key signature with an appended extra value, i.e., the hash pre-image.

The format changes are only a few new requirements for `authenticatedAttributes` and `unauthenticatedAttributes` of the `SignerInfor` field. In a SAS signature, `SignerInfor` is the same as in plain PKCS#7, except:

- `authenticatedAttributes`:
  this field is not OPTIONAL, but MANDATORY. It must contain, at a minimum, two more attributes aside from those set in PKCS#7:
  - SAS_issuer_sn: IssuerAndSerialNumber – specifies the SAS client's certificate by issuer name and issuer-specific serial number
  - SAS_signed_token_index: INTEGER – specifies the SAS client signed one-time signature index (counter)
  - SAS_signed_token_value: OCTET STRING – specifies the SAS client signed one-time public key

  Note that PKCS#7 requires `issuerAndSerialNumber` in `SignerInfo` to identify signer's key. In SAS, this corresponds to SEM's key. Therefore, we require another field `SAS_issuer_sn` to identify the user's SAS certificate containing the SAS root key. The signed token is not placed into `ContentInfo` so that the message digest handling is the same as with any other public key signature type. Moreover, the token can be extracted from PKCS#7 independently, if necessary.

- `unauthenticatedAttributes`:
  this field is not OPTIONAL, but MANDATORY. It must contain:
  - `SAS_preimage_token_value`: OCTET STRING – specifies the SAS user's one-time hash pre-image of the signed token specified in `SAS_signedtoken_value`. This attribute is unsigned. It is inserted by the user when the SEM's half-signature is received and verified.

Because of format compatibility, a SAS signature can be shipped as a normal PKCS#7 signature. However, the verification method is obviously different. The normal PKCS#7 verification routines can only verify the SEM half-signature (i.e., RSA public key signature).

The extra step in (light) verification of a SAS signature is the comparison of the hash of `SAS_preimage_token_value` and the `SAS_signed_token_value` assuming light verification is used. Otherwise, as described above, the verifier checks the validity of `SAS_signed_token_value` and

`SAS_signed_token_index` by computing the iterative hash and comparing the result with the SAS root key in the signer's SAS certificate.

The fact that two parties participate in signing result in a semantic issue when SAS signatures are used in conjunction with S/MIME. Most S/MIME applications enforce a policy requiring the sender of the message (as shown in the RFC822 `From:` field) to match the e-mail address in the signer certificate. Unfortunately, in SAS, the sender is the holder of the SAS certificate, e.g., alice@wonderland.com. Whereas, the "signer" is the SEM, e.g., sem@wonderland.com. Therefore, a SAS verifier should be aware of the presence of the unsigned attribute and use the proper email address in comparison.

A.2 SAS Certificate

To support SAS attributes, we extended X509v3 handling [21] in the popular Openssl library [22]. In addition to the usual X509v3 fields, a SAS certificate also certifies the following:

- `SASHashType`: `DigestAlgorithmIdentifier` – identifies the hash algorithm used in generating the hash chain;
- `SASPublicKeyIdentifier`: `OCTET STRING` – SAS root key in the hash-chain.
- `SASPublicKeyPara`: `INTEGER` – length of the hash-chain.
- `SASServerName`: `STRING` – SEM's host name. This field indicates the location of SEM and has no security meaning.
- `SASSerialNumber`: `INTEGER` – SEM's certificate serial number. (Here it is assumed that the SEM and the user share the same CA). Uniquely identifies SEM's certificate and the corresponding public key.

A.3 Eudora Plug-in Details

We implemented the SAS plug-in as two email translators defined in Eudora's plug-in API [19]. Specifically, SAS signing is a *Q4-Transmission* translator and SAS verification is an *On-Display* translator.

SAS signing translator is invoked when Eudora is ready to send email and is fed with the entire email message, including its MIME header. When SAS signature protocol terminates, the whole SAS signature in PKCS#7 format is appended to the email body as an attachment with the MIME subtype `''x.SAS-signature''`.

SAS verification translator is called when Eudora is about to display a SAS-signed email. As in traditional signature verification, a certificate chain must be at hand. Our plug-in allows users to specify the root CA certificate, assuming, of course, that the SEM and the SAS client share the same certificate issuer. It is easy to build a chain by extracting SEM and client's certificate from the PKCS#7 signature. In this implementation, we chose not to adopt opaque signing. If the signature is invalid, an error message window is popped up while the original email body is still displayed.

Since SAS signature verification is different from normal S/MIME, non-Eudora applications, like Netscape or Outlook, cannot verify it without a special verification program. We provide such a standalone

Appendix B: Related Work on Certificate Revocation

• **CRLs and Δ-CRLs:** Certificate Revocation Lists are the most common way to handle certificate revocation. The Validation Authority (VA) periodically posts a signed list of all revoked certificates. These lists are placed on designated servers called CRL distribution points. Since these lists can get quite long, a VA may alternatively post a signed Δ-CRL which only contains the list of revoked certificates since the last CRL was issued. When verifying a signature on a message, the verifier checks that, at the time that the signature was issued, the signer's certificate was not on the CRL.

• **OCSP:** The Online Certificate Status Protocol (OCSP) [1] improves on CRLs by avoiding the transmission of long CRLs to every user and by providing more timely revocation information. The VA sends back a signed response indicating whether the specified certificate is currently revoked. When verifying a signature, the verifier sends an OCSP (certificate status request) query to the VA to check if the enclosed certificate is currently valid. The VA answers with a signed response indicating the certificate's revocation status. Note that OCSP prevents one from implementing stronger semantics: it is impossible to ask an OCSP VA whether a certificate was valid at some time in the past.

• **Certificate Revocation Trees:** Kocher [23] suggested an improvement over OCSP. Since the VA is a global service, it must be sufficiently replicated to handle the load of all validation queries. This means the VA's signing key must be replicated across

many servers which is either insecure or expensive (VA servers typically use tamper-resistance to protect the VA's signing key). Kocher's idea is to have a single highly secure VA periodically post a signed CRL-like data structure to many insecure VA servers. Users then query these insecure VA servers. The data structure (CRT) proposed by Kocher is a hash tree where the leaves are the currently revoked certificates sorted by serial number The root of the hash tree is signed by the VA.

A user wishing to validate a certificate issues a query to the closest VA server. Any insecure VA can produce a convincing proof that the certificate is (or is not) on the CRT. If n certificates are currently revoked, the length of the proof is $O(\log n)$. In contrast, the length of the validity proof in OCSP is $O(1)$.

• **Skip-lists and 2-3 trees:** One problem with CRTs is that, every time a certificate is revoked, the entire CRT must be recomputed and distributed in its entirety to the various VA servers. A data structure allowing for dynamic updates would solve this problem since the secure VA would only need to send small updates to the data structure along with a signature on the new root of the structure. Both 2-3 trees proposed by Naor and Nissim [24] and skip-lists proposed by Goodrich [25] are natural data structures for this purpose. Additional data structures were proposed in [26]. When a total of n certificates are already revoked and k new certificates must be revoked during the current time period, the size of the update message to the VA servers is $O(k \log n)$ (as opposed to $O(n)$ with CRT's). The proof of certificate's validity is $O(\log n)$, same as with CRTs.

A Method for Fast Revocation of Public Key Certificates and Security Capabilities*

Dan Boneh† dabo@cs.stanford.edu
Xuhua Ding‡ xhding@isi.edu
Gene Tsudik‡ gts@ics.uci.edu
Chi Ming Wong† bc@cs.stanford.edu

Abstract

We present a new approach to fast certificate revocation centered around the concept of an on-line semi-trusted mediator (SEM). The use of a SEM in conjunction with a simple threshold variant of the RSA cryptosystem (mediated RSA) offers a number of practical advantages over current revocation techniques. Our approach simplifies validation of digital signatures and enables certificate revocation within legacy systems. It also provides immediate revocation of all security capabilities. This paper discusses both the architecture and implementation of our approach as well as performance and compatibility with the existing infrastructure. Our results show that threshold cryptography is practical for certificate revocation.

1 Introduction

We begin this paper with an example to illustrate the premise for this work. Consider an organization – industrial, government or military – where all employees (referred to as *users*) have certain authorities and authorizations. We assume that a modern Public Key Infrastructure (PKI) is available and all users have digital signature, as well as encryption, capabilities. In the course of performing routine everyday tasks users take advantage of secure applications such as email, file transfer, remote log-in and web browsing.

Now suppose that a trusted user (Alice) does something that warrants immediate revocation of her security privileges. For example, Alice might be fired, or she may suspect that her private key has been compromised. Ideally, immediately following revocation, Alice should be unable to perform any security operations and use any secure applications. Specifically, this means:

- Alice cannot read secure (private) email. This includes encrypted email that is already residing on Alice's email server. Although encrypted email may be basically delivered (to Alice's email server), she cannot decrypt it.
- Alice cannot generate valid digital signatures on any further messages. (However, signatures generated by Alice prior to revocation may need to remain valid.)
- Alice cannot authenticate herself to corporate servers.

In Section 7, we discuss current revocation techniques and demonstrate that the above requirements are impossible to satisfy with these techniques. Most importantly, current techniques do not provide immediate revocation.

1.1 The SEM architecture.

Our approach to immediate revocation of security capabilities is called the SEM architecture. It is easy to use and its presence is transparent to peer users (those that encrypt messages and verify signatures). The basic idea is as follows:

We introduce a new entity, referred to as a SEM *(SEcurity Mediator). A* SEM *is an online semi-trusted server. To sign or decrypt a message, Alice must first obtain a message-specific token from the* SEM*. Without this token Alice cannot use her private key.*[1] *To revoke Alice's ability to sign or de-*

*This work is supported by the Defense Advanced Project Agency (DARPA) under contract F30602-99-1-0530.

†Computer Science Department, Stanford University.
‡Department of Information and Computer Science, University of California, Irvine.

[1] The exact description of the token is in Section 2.

crypt, the security administrator instructs the SEM *to stop issuing tokens for Alice's public key. At that instant, Alice's signature and/or decryption capabilities are revoked. For scalability reasons, a* SEM *serves many users.*

We emphasize that the SEM architecture is transparent to peer users: with SEM's help, Alice can generate a standard RSA signature, and decrypt standard messages encrypted with her RSA public key. Without SEM's help, she cannot perform either of these operations. The SEM architecture is implemented using threshold RSA [3] as described in section 2.

To experiment with this architecture we implemented it using OpenSSL [12]. SEM is implemented as a daemon process running on a server. We describe our implementation, the protocols used to communicate with the SEM, and give performance results in Sections 5 and 6.

We also built a plug-in for the Eudora client enabling users to send signed email. All signatures are generated with SEM's help (see [15]). Consequently, signing capabilities can be easily revoked.

1.2 Decryption and signing in the SEM architecture

We now describe in more detail how decryption and signing is done in the SEM architecture:

– Decryption: suppose Alice wishes to decrypt an email message using her private key. Recall that encrypted email is composed of two parts: (1) a short header containing a message-key encrypted using Alice's public key, and (2) the body contains the email message encrypted using the message-key. To decrypt, Alice first sends the short header to her SEM. SEM responds with a short token. This token enables Alice to read her email. However, it contains no useful information to anyone but Alice. Hence, communication with the SEM does not have to be protected or authenticated. We note that interaction with the SEM is fully managed by Alice's email reader and does not require any intervention on Alice's part. This interaction does not use Alice's private key. If Alice wants to read her email offline, the interaction with the SEM takes places at the time Alice's email client downloads Alice's email from the email server.

– Signatures: suppose Alice wishes to sign a message using her private key. She sends a hash of the message to the SEM which, in turn, responds with a short token enabling Alice to generate the signature. As with decryption, this token contains no useful information to anyone but Alice; therefore, the interaction with the SEM is not encrypted or authenticated.

Note that all interaction with the SEM involves very short messages.

1.3 Other benefits of using a SEM

Our initial motivation for introducing a SEM is to enable immediate revocation of Alice's key. We point out that the SEM architecture provides two additional benefits over standard revocation techniques: (1) simplified signature validation, and (2) enabling revocation in legacy systems. These benefits apply when the following semantics for validating digital signatures are used:

Binding signature semantics: a digital signature is considered valid if the certificate associated with the signature was valid at the time the signature was issued.

A consequence of binding signature semantics is that all signatures issued prior to certificate revocation are valid. Binding semantics are natural in business contracts. For example, suppose Alice and Bob enter into a contract. They both sign the contract at time T. Bob begins to fulfill the contract and incurs certain costs in the process. Now, suppose at time $T' > T$, Alice revokes her own certificate. Is the contract valid at time T'? Using binding semantics, Alice is still bound to the contract since it was signed at time T when her certificate was still valid. In other words, Alice cannot nullify the contract by causing her own certificate to be revoked.

(We note that binding semantics are inappropriate in some scenarios. For example, if a certificate is obtained from a CA under false pretense, e.g., Alice masquerading as Bob, the CA should be allowed to declare at any time that **all** signatures ever issued under that certificate are invalid.)

Implementing binding signature semantics with existing revocation techniques is complicated, as discussed in Section 7. Whenever Bob verifies a signa-

ture generated by Alice, Bob must also verify that Alice's certificate was valid at the time the signature was issued. In fact, every verifier of Alice's signature must perform this certificate validation step. However, unless a trusted timestamping service is involved in generating all of Alice's signatures, Bob cannot trust the timestamp provided by Alice in her signatures.

Implementing binding semantics with the SEM architecture is trivial. To validate Alice's signature, a verifier need only verify the signature itself. There is no need to check the status of Alice's certificate.[2] Indeed, once Alice's certificate is revoked she can no longer generate valid signatures. Therefore, the mere existence of the signature implies that Alices's certificate was valid at the time the signature was issued.

The above discussion brings out two additional benefits of a SEM over existing revocation techniques, assuming binding semantics are sufficient.

– Simplified signature validation. Verifiers need not validate the signer's certificate. The existence of a (verifiable) signature is, in itself, a proof of signature's validity.

– Enabling revocation in legacy systems. Consider legacy systems doing signature verification. Often, such systems have no certificate validation capabilities. For example, old browsers (e.g., Netscape 3.0) verify server certificates without any means for checking certificate revocation status. In SEM architecture, certificate revocation is provided without any change to the verification process in these legacy systems. (The only aspect that needs changing is the signature generation process. However, we note that, often, only a few entities generate signatures, e.g., CAs and servers.)

2 Mediated RSA

We now describe in detail how the SEM interacts with users to generate tokens. The proposed SEM architecture is based on a variant of RSA which we call Mediated RSA (mRSA). The main idea in mRSA is to split each RSA private key into two parts using threshold RSA [3]. One part is given to a user while the other is given to a SEM. If the user and the SEM cooperate, they employ their respective half-keys in a way that is functionally equivalent to (and indistinguishable from) standard RSA. The fact that the private key is not held in its entirety by any one party is transparent to the outside world, i.e., to the those who use the corresponding public key. Also, knowledge of a half-key cannot be used to derive the entire private key. Therefore, neither the user nor the SEM can decrypt or sign a message without mutual consent. (A single SEM serves a multitude of users.)

2.1 mRSA in detail

Public Key. As in RSA, each user (U_i) has a public key $EK_i = (n_i, e_i)$ where the modulus n_i is product of two large primes p_i and q_i and e_i is an integer relatively prime to $\phi(n_i)$.

Secret Key. As in RSA, there exists a corresponding secret key $DK_i = (n_i, d_i)$ where $d_i * e_i = 1 \pmod{\phi(n_i)}$. However, as mentioned above, no one has possession of d_i. Instead, d_i is effectively split into two parts d_i^u and d_i^{sem} which are held by the user U_i and a SEM, respectively. The relationship among them is:

$$d_i = d_i^{sem} + d_i^u \mod \phi(n)$$

mRSA Key Setup. Recall that, in RSA, each user generates its own modulus n_i and a public/secret key-pair. In mRSA, a trusted party (most likely, a CA) takes care of all key setup. In particular, it generates a distinct set: $\{p_i,\ q_i,\ e_i,\ \text{and}\ d_i, d_i^{sem}\}$ for each user. The first four are generated in the same manner as in standard RSA. The fifth value, d_i^{sem}, is a random integer in the interval $[1, n_i]$. The last value is set as: $d_i^u = d_i - d_i^{sem}$.

After CA computes the above values, d_i^{sem} is securely communicated to a SEM and d_i^u – to the user U_i. The details of this step are elaborated in Section 5.

mRSA Signatures. A user generates a signature on a message m as follows:

[2] We are assuming here that revocation of Alice's key is equavalent to revocation of Alice's certificate. In general, however, Alice's certificate may encode many rights, not just the right to use her key(s). It is then possible to revoke only some of these rights while not revoking the entire certificate.

1. The user U_i first sends a hash of the message m to the appropriate SEM.
   - SEM checks that U_i is not revoked and, if so, computes a partial signature $PS_{sem} = m^{d_i^{sem}} \pmod{n_i}$ and replies with it to the user. This PS_{sem} is the token enabling signature generation.
   - concurrently, U_i computes $PS_u = m^{d_i^u} \pmod{n_i}$
2. U_i receives PS_{sem} and computes $m' = (PS_{sem} * PS_u)^{e_i} \pmod{n_i}$. If $m' = m$, the signature is set to: $(PS_{sem} * PS_u) = m^{d_i} \pmod{n_i}$.

Note that in Step 2 the user U_i validates the response from the SEM. Signature verification is identical to that in standard RSA.

mRSA Encryption. The encryption process is identical to that in standard RSA. (In other words, ciphertext is computed as $c = m^{e_i} \pmod{n_i}$ where m is an appropriately padded plaintext, e.g., using OAEP.) Decryption, on the other hand, is very similar to signature generation above.

1. upon obtaining an encrypted message c, user U_i sends it to the appropriate SEM.
   - SEM checks that U_i is not revoked and, if so, computes a partial cleartext $PC_{sem} = c^{d_i^{sem}} \pmod{n_i}$ and replies to the user.
   - concurrently, U_i computes $PC_u = c^{d_i^u} \pmod{n_i}$.
2. U_i receives PC_{sem} and computes $c' = (PC_{sem} * PC_u)^{e_i} \pmod{n_i}$. If $c' = c$, the cleartext message is: $(PC_{sem} * PC_u) = c^{d_i^u}$.

2.2 Notable Features

As mentioned earlier, mRSA is only a slight modification of the RSA cryptosystem. However, at a higher, more systems level, mRSA affords some interesting features.

CA-based Key Generation. Recall that, in RSA, a private/public key-pair is typically generated by its intended owner. In mRSA the key-pair is typically generated by a CA, implying that the CA knows the private keys belonging to all users. In the global Internet this is clearly undesirable. However, in a medium-sized organization this "feature" provides key escrow. For example, if Alice is fired, the organization can still access her work-related files by obtaining her private key from the CA.

If key escrow is undesirable, it is easy to extend the system in a way that no entity ever knows Alice's private key (not even Alice or the CA). To do so, we can use a technique due to Boneh and Franklin [2] to generate an RSA key-pair so that the private key is shared by a number of parties since its creation (see also [4]). This technique has been implemented in [8]. It can be used to generate a shared RSA key between Alice and the SEM so that no one knows the full private key. Our initial implementation does not use this method. Instead, the CA does the full key setup.

Immediate Revocation. The notoriously difficult revocation problem is greatly simplified in mRSA. In order to revoke a user's public key, it suffices to notify that user's SEM. Each SEM merely maintains a list of revoked users which is consulted upon every service request. Our implementation uses standard X.509 Certificate Revocation Lists (CRL's) for this purpose.

Transparency. mRSA is completely transparent to those who encrypt data for mRSA users and those who verify signatures produced by mRSA users. To them, mRSA appears indistinguishable from standard RSA. Furthermore, mRSA certificates are identical to standard RSA certificates.

Coexistence. mRSA's built-in revocation approach can co-exist with the traditional, explicit revocation approaches. For example, a CRL- or a CRT-based scheme can be used in conjunction with mRSA in order to accommodate existing implementations that require verifiers (and encryptors) to perform certificate revocation checks.

CA Communication. CA remains an off-line entity. mRSA certificates, along with private half-keys are distributed to the user and SEM-s in an off-line manner. This follows the common certificate issuance and distribution paradigm. In fact, in our implementation (Section 5) there is no need for the CA and the SEM to ever communicate directly.

No Authentication. mRSA does not require any explicit authentication between a SEM and a user. Instead, a user implicitly authenticates a SEM by verifying its own signature (or encryption) as described in Section 2.1. In fact, signature and encryption verification steps assure the user of the integrity of the communication with the SEM.

3 Architecture

The overall architecture is made up of three components: CA, SEM, and user.

A single CA governs a (small) number of SEMs. Each SEM, in turn, serves many users. The assignment of users to SEMs is assumed to be handled off-line by a security administrator. A user may be served by multiple SEM's.

Our CA component is a simple add-on to the existing CA and is thus considered an off-line entity. For each user, the CA component takes care of generating an RSA public key, a corresponding RSA public key certificate and a pair of half-keys (one for the user and one for the SEM) which, when combined, form the RSA private key. The respective half-keys are then delivered, out-of-band, to the interested parties.

The user component consists of the client library that provides the mRSA sign and mRSA decrypt operations. (As mentioned earlier, the verify and encrypt operations are identical to standard RSA.) It also handles the installation of the user's credentials at the local host.

The SEM component is the critical part of the architecture. Since a single SEM serves many users, performance, fault-tolerance and physical security are of paramount concern. The SEM is basically a daemon process that processes requests from its constituent users. For each request, SEM consults its revocation list and refuses to help sign (or decrypt) for any revoked users. A SEM can be configured to operate in a stateful or stateless model. The former involves storing per user state (half-key and certificate) while, in the latter, no per user state is kept, however, some extra processing is incurred for each user request. The tradeoff is fairly clear: per user state and fast request handling versus no state and somewhat slower request handling.

We now describe the SEM architecture in more detail. A user's request is initially handled by the SEM controller where the packet format is checked. Next, the request is passed on to the client manager which performs a revocation check. If the requesting user is not revoked, the request is handled depending on the SEM state model. If the SEM is stateless, it expects to find the so-called SEM *bundle* in the request. This bundle, as discussed in more detail later, contains the mRSA half-key, d_i^{SEM}, encrypted (for the SEM, using its public key) and signed (by the CA). The bundle also contains the RSA public key certificate for the requesting user. Once the bundle is verified, the request is handled by either the $\mathsf{mRSA}_{\mathsf{sign}}$ or $\mathsf{mRSA}_{\mathsf{decrypt}}$ component. In case of the appropriate signature request, the optional timestamping service is invoked. If the SEM maintains user state, the bundle is expected only in the initial request. The same process as above is followed, however, the SEM's half-key and the user's certificate are stored locally. In subsequent user requests, the bundle (if present) is ignored and local state is used instead.

The administrator communicates with the SEM via the admin interface. The interface enables the administrator to manipulate the revocation list.

4 Security of the SEM architecture

We now briefly summarize the security features of mRSA and the SEM architecture.

First, consider an attacker trying to subvert a user (Alice). The attacker's goal is to decrypt a message sent to Alice or to forge Alice's signature on a certain message. Recall that the token sent back to Alice is $t = x^{d^{sem}} \bmod N$ for some value of x. The attacker sees both x and the token t. In fact, since there is no authentication of the user's request to the SEM, the attacker can obtain this t for any x of its choice. We claim that this information is of no use to an attacker. After all, d^{sem} is just a random number in $[1, n]$ independent of the rest of the attacker's view. More precisely, we argue that any attack possible with the SEM architecture is also possible when the user uses standard RSA. This statement can be proven using a simulation argument. In attacking standard RSA one can simulate the SEM (by picking a random integer d^{sem} in $[1, n]$) and thus use the attack on the SEM to mount an attack on standard

RSA. Furthermore, the attacker cannot masquerade as the SEM since Alice checks all responses from the SEM as described in Section 2.1.

Suppose the attacker is able to compromise the SEM and expose the secret key d^{sem}. This enables the attacker to "unrevoke" revoked, or block possible future revocation of currently valid, certificates. However, knowledge of d^{sem} does not enable the attacker to decrypt messages or sign messages on behalf of users. Nevertheless, it is desirable to protect the SEM's key. A standard approach is to distribute the key among a number of SEM servers using secret sharing. Furthermore, the key should never be reconstructed at a single location. To extract the SEM's key an attacker would need to break into multiple SEM servers. When using mRSA, it is possible to distribute the SEM's secret in this way using standard techniques from threshold cryptography [3].

Once Alice's key is revoked, she cannot decrypt or sign messages using her private key. To show this, we argue that, if Alice could sign or decrypt messages using only her share of private key, then RSA is insecure.

Finally, note that each user is given her own random RSA modulus n_i. This means that if a number of users are compromised (or a number of users collude) there is no danger to other users. The private keys of the compromised users will be exposed, but private keys of all other users will remain unaffected.

5 Implementation

We implemented the entire SEM architecture for the purposes of experimentation and validation. The reference implementation is publicly available. Following the architecture described earlier, the implementation is composed of three parts:

1. CA and Admin Utilities:
   includes certificate issuance and revocation interface.

2. SEM daemon:
   SEM architecture as described in Section 3

3. Client libraries:
   mRSA user functions accessible via an API.

Our reference implementation uses the popular OpenSSL [12] library as the low-level cryptographic platform. OpenSSL incorporates a multitude of cryptographic functions and large-number arithmetic primitives. In addition to being efficient and available on many common hardware and software platforms, OpenSSL adheres to the common PKCS standards and is in the public domain.

The SEM daemon and the CA/Admin utilities are implemented on Linux and Solaris while the client libraries are available on both Linux and Windows98 platforms.

In the initialization phase, CA utilities are used to set up the RSA public key-pair for each client (user). The set up process follows the description in Section 2. Once the mRSA parameters are generated, two structures are exported: 1) SEM *bundle*, which includes the SEM's half-key d_i^{SEM}, and 2) user *bundle*, which includes d_i^u and the entire server bundle. The format of both SEM and user bundles conforms to the PKCS#7 standard.

The server bundle is basically an RSA envelope signed by the CA and encrypted with the SEM's public key. The client bundle is a shared-key envelope also signed by the CA and encrypted with the user-supplied key which can be a password or a passphrase. (A user cannot be assumed to have a pre-existing public key.)

After issuance, the user bundle is distributed in an out-of-band manner to the appropriate user. Before attempting any mRSA transactions, the user must first decrypt and verify the bundle. A separate utility program is provided for this purpose. With it, the bundle is decrypted with the user-supplied key, the CA's signature is verified, and, finally, the user's new certificate and half-key are extracted and stored locally.

To sign or decrypt a message, the user starts with sending an mRSA request with the SEM bundle piggybacked. The SEM processes the request and the bundle contained therein as described in Section 3. (Recall that the SEM bundle is processed based on the state model of the particular SEM.) All mRSA packets have a common packet header; the payload format depends on the packet type. The packet header is defined in Figure 1.

```
0              8               16             24              32
+--------------+---------------+--------------+---------------+
| PROTOCOL     |  PACKET_TYPE  |     DATA_LENGTH              |
+--------------+---------------+--------------+---------------+

PROTOCOL   :  protocol identifier. Set to MRSA(=1) in current code
PACKET_TYPE:  one of the following:
         1) REG_REQ_T : register request
         2) REG_RLY_T : register reply
         3) SIG_REQ_T : signature request
         4) SIG_RLY_T : signature reply
         5) DEC_REQ_T : decrypt request
         6) DEC_RLY_T : decrypt reply
```

Figure 1: mRSA Packet Header

5.1 Email client plug-in

To demonstrate the ease of using the SEM architecture we implemented a plug-in for the Eudora email reader [15]. When sending signed email the plug-in reads the user bundle described in the previous section. It obtains the SEM address from the bundle and then communicates with the SEM to sign the email. The resulting signed email can be verified using any S/MIME capable email client such as Microsoft Outlook. In other words, the email recipient is oblivious to the fact that a SEM is used to control the sender's signing capabilities.

Figure 2 shows a screen snap shot of trying to send signed email using a revoked key. In this example, the plug-in contacts the SEM and is told that the SEM will not supply the token for a revoked key. Consequently, the plug-in displays a message informing the user that the email cannot be signed.

6 Experimental Results

We conducted a number of experiments in order to evaluate the practicality of the proposed architecture and our implementation.

We ran the SEM daemon on a Linux PC equipped with an 800 Mhz Pentium III processor. Two different clients were used. The fast client was on another Linux PC with a 930 MHz Pentium III. Both SEM and fast client PC-s had 256M of RAM. The slow client was on a Linux PC with 466 MHz Pentium II and 128M of RAM. Although an 800 Mhz processor is not exactly state-of-the-art, we opted to err on the side of safety and assume a relatively conservative (i.e., slow) SEM platform. In practice, a SEM might reside on much faster hardware and is likely to be assisted by an RSA hardware acceleration card.

Each experiment involved one thousand iterations. All reported timings are in milliseconds (rounded to the nearest 0.1 *ms*). The SEM and client PCs were located in different sites interconnected by a high-speed regional network. All protocol messages are transmitted over UDP.

Client RSA key (modulus) sizes were varied among 512, 1024 and 2048 bits. (Though it is clear that 512 is not a realistic RSA key size any longer.) The timings are only for the mRSA sign operation since mRSA decrypt is operationally almost identical.

6.1 Communication Overhead

In order to gain precise understanding of our results, we first provide separate measurements for communication latency in mRSA. Recall that both mRSA operations involve a request from a client followed by a reply from a SEM. As mentioned above, the test PCs were connected by a high-speed regional network. We measured communication latency by varying the key size which directly influences message sizes. The results are shown in Table 1 (message sizes are in bytes). Latency is calculated as the round-trip delay between the client and the SEM. The numbers are identical for both client types.

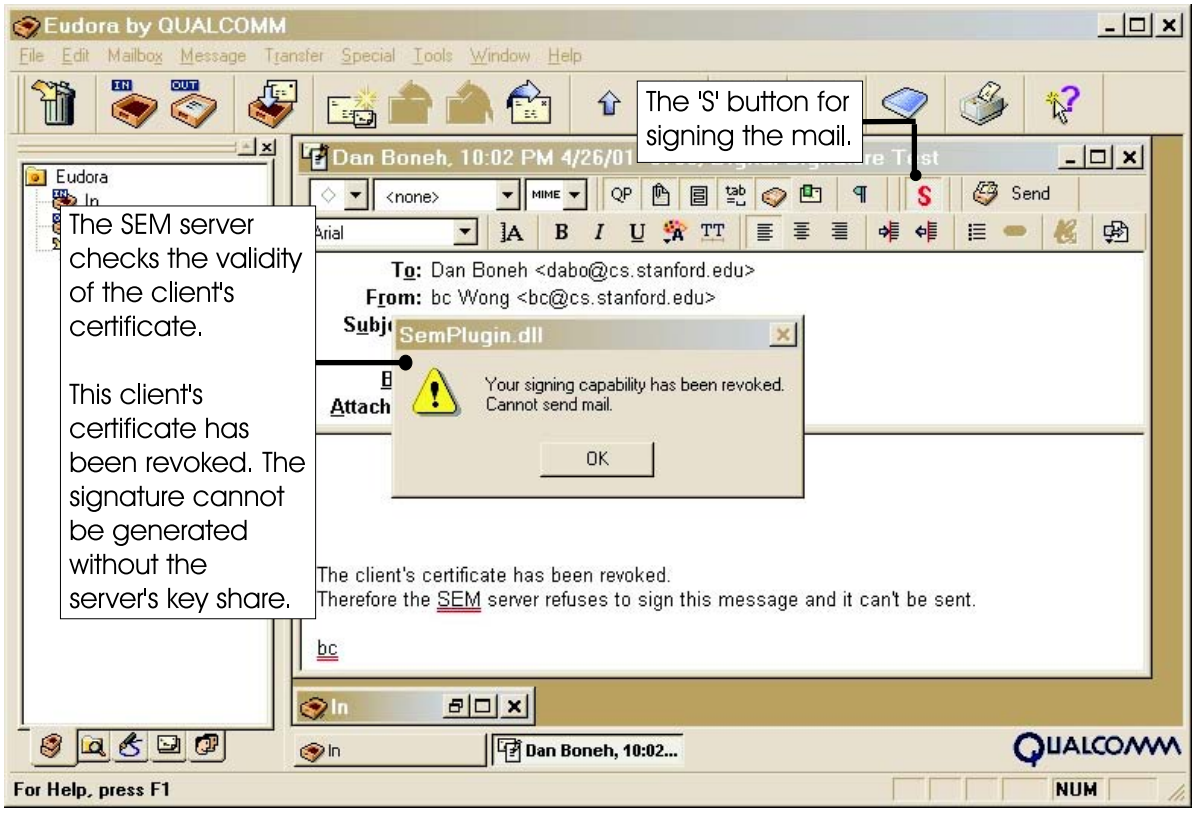

Figure 2: Screen snapshot of SEM email plug-in

Keysize (bits)	Message Size (bytes)	Comm. latency (ms)
512	102	4.0
1024	167	4.5
2048	296	5.5

Table 1: Communication latency

6.2 Standard RSA

As a point of comparison, we initially timed the standard RSA sign operation in OpenSSL (Version 0.9.6) with three different key sizes on each of our three test PCs. The results are shown in Tables 2 and 3. Each timing includes a message hash computation followed by an exponentiation. Table 2 reflects optimized RSA computation where the *Chinese Remainder Theorem (CRT)* is used to speed up exponentiation (essentially exponentiation is done modulo the prime factors rather than modulo N). Table 3 reflects unoptimized RSA computation without the benefit of the CRT. Taking advantage of the CRT requires knowledge of the factors (p and q) of the modulus n. Recall that, in mRSA, neither the SEM nor the user know the factorization of the modulus, hence, with regard to its computation cost, mRSA is more akin to unoptimized RSA.

As evident from the two tables, the optimized RSA performs a factor of 3-3.5 faster for the 1024- and 2048-bit moduli than the unoptimized version. For 512-bit keys, the difference is slightly less marked.

6.3 mRSA Measurements

The mRSA results are obtained by measuring the time starting with the message hash computation by the user (client) and ending with the verification of the signature by the user. The measurements are

Keysize (bits)	466 Mhz PII (slow client)	800 Mhz PIII (SEM)	930 Mhz PIII (fast client)
512	2.9	1.4	1.4
1024	14.3	7.7	7.2
2048	85.7	49.4	42.8

Table 2: RSA results with CRT (in milliseconds).

Keysize (bits)	466 Mhz PII (slow client)	800 Mhz PIII (SEM)	930 Mhz PIII (fast client)
512	6.9	4.0	3.4
1024	43.1	24.8	21.2
2048	297.7	169.2	144.7

Table 3: Standard RSA results without CRT (in milliseconds).

illustrated in Table 4.

It comes as no surprise that the numbers for the slow client in Table 4 are very close to the unoptimized RSA measurements in Table 3. This is because the time for an mRSA operation is determined solely by the client for 1024- and 2048- bit keys. With a 512-bit key, the slow client is fast enough to compute its PS_u in $6.9ms$. This is still under $8.0ms$ (the sum of $4ms$ round-trip delay and $4ms$ RSA operation at the SEM).

The situation is very different with a fast client. Here, for all key sizes, the timing is determined by the sum of the round-trip client-SEM packet delay and the service time at the SEM. For instance, $178.3ms$ (clocked for 2048-bit keys) is very close to $174.7ms$ which is the sum of $5.5ms$ communication delay and $169.2ms$ unoptimized RSA operation at the SEM.

All of the above measurements were taken with the SEM operating in a stateful mode. In a stateless mode, SEM incurs further overhead due to the processing of the SEM bundle for each incoming request. This includes decryption of the bundle and verification of the CA's signature found inside. To get an idea of the mRSA overhead with a stateless SEM, we conclude the experiments with Table 5 showing the bundle processing overhead. Only 1024- and 2048-bit SEM key size was considered. (512-bit keys are certainly inappropriate for a SEM.) The CA key size was constant at 1024 bits.

7 Comparison of SEM with existing certificate revocation techniques

Certificate revocation is a well recognized problem with the existing Public Key Infrastructure (PKI). Several proposals address this problem. We briefly review these proposals and compare them to the SEM architecture. For each proposal we describe how it applies to signatures and to encryption. For simplicity we use signed and encrypted Email as an example application. We refer to the entity validating and revoking certificates as the Validation Authority (VA). Typically, the VA is the same entity as the Certificate Authority (CA). However, in some cases these are separate organizations.

A note on timestamping. Binding signature semantics (Section 1.3) for signature verification states that a signature is considered valid if the key used to generate the signature was valid at the time signature generation. Consequently, a verifier must establish exactly when a signature was generated. Hence, when signing a message, the signer must interact with a trusted timestamping service to obtain a trusted timestamp and a signature over the user's (signed) message. This proves to any verifier that a signature was generated at a specific time. All the techniques discussed below require a signature to contain a timestamp indicating when a signature was issued. We implicitly assume this service. As we will see, there is no need for a trusted time service to implement binding signature semantics with the SEM architecture.

Keysize (bits)	466 Mhz PII (slow client)	930 Mhz PIII (fast client)
512	8.0	9.9
1024	45.6	31.2
2048	335.6	178.3

Table 4: Timings for mRSA (in milliseconds).

SEM key size	Bundle overhead
1024	8.1
2048	50.3

Table 5: Bundle overhead in mRSA with a SEM in a stateless mode (in milliseconds).

7.1 Review of existing revocation techniques

CRLs and Δ-CRLs: Certificate Revocation Lists are the most common way to handle certificate revocation. The Validation Authority (VA) periodically posts a signed list of all revoked certificates. These lists are placed on designated servers called CRL distribution points. Since these lists can get quite long, the VA may alternatively post a signed Δ-CRL which only contains the list of revoked certificates since the last CRL was issued. For completeness, we briefly explain how CRLs are used in the context of signatures and encryption:

- Encryption: at the time email is sent, the sender checks that the receiver's certificate is not on the current CRL. The sender then sends encrypted email to the receiver.
- Signatures: when verifying a signature on a message, the verifier checks that, at the time that the signature was issued, the signer's certificate was not on the CRL.

OCSP: The Online Certificate Status Protocol (OCSP) [11] improves on CRLs by avoiding the transmission of long CRLs to every user and by providing more timely revocation information. To validate a specific certificate in OCSP, the user sends a certificate status request to the VA. The VA sends back a signed response indicating whether the specified certificate is currently revoked. OCSP is used as follows for Encryption and signatures:

- Signatures: When verifying a signature, the verifier sends an OCSP query to the VA to check if the corresponding certificate is currently valid. Note that the current OCSP protocol prevents one from implementing binding semantics: it is not possible to ask an OCSP responder whether a certificate was valid at some time in the past. Hopefully this will be corrected in future versions of the protocol.

  One could potentially abuse the OCSP protocol and provide binding semantics as follows. To sign a message, the signer generates the signature, and also sends an OCSP query to the VA. The VA responds with a signed message saying that the certificate is currently valid. The signer appends both the signature and the response from the VA to the message. To verify the signature, the verifier checks the VA's signature on the validation response. The response from the VA provides a proof that the signer's certificate is currently valid. This method reduces the load on the VA: it is not necessary to contact the VA every time a signature is verified. Unfortunately, there is currently no infrastructure to support this mechanism.
- Encryption: Every time the sender sends an encrypted message to the receiver she sends an OCSP query to the VA to ensure that the receiver's certificate is still valid.

Certificate Revocation Trees: Kocher suggested an improvement over OCSP [7]. Since the VA is a global service it must be sufficiently replicated in order to handle the load of all the validation queries. This means the VA's signing key must be replicated across many servers which is either insecure or expensive (VA servers typically use tamper-resistance to protect the VA's signing key). Kocher's idea is to have a single highly secure VA periodically post a signed CRL-like data structure to many insecure VA servers. Users then query these insecure VA servers. The data structure proposed by Kocher is a hash tree where the leaves are the currently revoked certificates sorted by serial number (lowest serial num-

ber is the left most leaf and the highest serial number is the right most leaf). The root of the hash tree is signed by the VA. This hash tree data structure is called a Certificate Revocation Tree (CRT).

When a user wishes to validate a certificate CERT she issues a query to the closest VA server. Any insecure VA can produce a convincing proof that CERT is (or is not) on the CRT. If n certificates are currently revoked, the length of the proof is $O(\log n)$. In contrast, the length of the validity proof in OCSP is $O(1)$.

Skip-lists and 2-3 trees: One problem with CRT's is that, every time a certificate is revoked, the entire CRT must be recomputed and distributed in its entirety to the various VA servers. A data structure allowing for dynamic updates would solve this problem since the secure VA would only need to send small updates to the data structure along with a signature on the new root of the structure. Both 2-3 trees proposed by Naor and Nissim [10] and skip-lists proposed by Goodrich [5] are natural data structures for this purpose. Additional data structures were proposed in [1]. When a total of n certificates are already revoked and k new certificates must be revoked during the current time period, the size of the update message to the VA servers is $O(k \log n)$ (as opposed to $O(n)$ with CRT's). The proof of certificate's validity is $O(\log n)$, same as with CRTs.

7.2 Comparison with SEM architecture

CRLs and OCSP are the most commonly deployed certificate revocation techniques. Some positive experiments with skip-lists are reported in [5]. We compare the SEM architecture with CRLs and OCSP. Since CRT's and skip-lists are used in the same way as OCSP (i.e., query a VA to obtain a proof of validity) most everything in our OCSP discussion applies to these methods as well.

Immediate revocation: Suppose we use CRLs for revocation. Then, Bob verifies a signature or encrypts a message he must first download a long CRL and verify that the Alice's certificate is not on the CRL. Note that Bob is uninterested in all but one certificate on the CRL. Nevertheless, he must download the entire CRL since, otherwise, the VA's signature on the CRL cannot be verified. Since CRLs and Δ-CRLs tend to get long, they are downloaded infrequently, e.g., once a week or month. As a result, certificate revocation might only take effect a month after the revocation occurs. The SEM architecture solves this problem altogether.

Suppose now that OCSP is usd for revocation. Whenever Bob sends email to Alice he first issues an OCSP query to verify validity of Alice's certificate. He then sends email encrypted with Alice's public key. The encrypted email could sit on Alice's email server for a few hours or days. If, during this time, Alice's key is revoked (e.g., because Alice is fired or looses her private key) there is nothing preventing the holder of Alice's private key from decrypting the email after revocation. The SEM solves this problem by disabling the private key immediately after revocation.

Implicit timestamping: Both OCSP and CRLs require the signer to contact a trusted time service at signature generation time to obtain a secure timestamp for the signature. Otherwise, a verifier cannot determine with certainty when the signature was issued. If binding semantics are sufficient, the time service is unnecessary when using the SEM architecture. Once a certificate is revoked, the corresponding private key can no longer be used to issue signatures. Therefore, a verifier holding a signature is explicitly assured that the signer's certificate was valid at the time the signature was generated.

Shifted validation burden: With current PKIs, the burden of validating certificates is placed on: (1) senders of encrypted messages and (2) verifiers of signed messages. In the SEM architecture, the burden of certificate validation is reversed: (1) receivers of encrypted messages and (2) signers (generators) of signed messages.

SEM **Replication (A disadvantage):** Since many users need to use the SEM for decryption and signing, it is natural to replicate it. However, replicating the SEM across organizations is not recommended for the same reason that replicating the VA in OCSP is not recommended. Essentially, the SEM generates tokens using a private key known only to the SEM. The result of exposing this key is that an attacker could unrevoke certificates. Replicating the SEM might make it easier to expose the SEM's key. Hence, the SEM architecture is mainly applicable in the same environments where OCSP is used, i.e., mainly medium-sized organizations. The SEM architecture is not geared towards the global Internet.

8 Conclusions

We described a new approach to certificate revocation. Rather than revoking the user's certificate our approach revokes the user's ability to perform cryptographic operations such as signature generation and decryption. This approach has several advantages over traditional certificate revocation techniques: (1) revocation is instantaneous – the instant the user's certificate is revoked the user can no longer decrypt or sign messages, (2) when using binding signature semantics there is no need to validate the signer's certificate during signature verification, and (3) using mRSA this revocation technique is transparent to the peer – the system generates standard RSA signatures and decrypts standards RSA encrypted messages.

We implemented the SEM architecture for experimentation purposes. Our measurements of the implementation show that signature and decryption times are essentially unchanged from the user's perspective. Therefore, we believe this architecture is appropriate for a medium-size organization where tight control of security capabilities is desired. The SEM architecture is not designed for the global Internet.

References

[1] W. Aiello, S. Lodha, R. Ostrovsky, "Fast digital identity revocation", In proceedings of CRYPTO '98.

[2] D. Boneh, M Franklin, "Efficient generation of shared RSA keys", In Proceedings of Crypto' 97, Lecture Notes in Computer Science, Vol. 1233, Springer-Verlag, pp. 425–439, 1997.

[3] P. Gemmel, "An introduction to threshold cryptography", in CryptoBytes, a technical newsletter of RSA Laboratories, Vol. 2, No. 7, 1997.

[4] N. Gilboa, "Two Party RSA Key Generation", in Proceedings of Crypto '99.

[5] M. Goodrich, R. Tamassia, and A. Schwerin, "Implementation of an Authenticated Dictionary with Skip Lists and Commutative Hashing", In Proceedings of DARPA DISCEX II, June 2001.

[6] S. Haber, W.S. Stornetta, "How to timestamp a digital document", J. of Cryptology, Vol. 3, pp. 99–111, 1991.

[7] P. Kocher, "On Certificate Revocation and Validation", Financial Cryptography – FC '98, Lecture Notes in Computer Science, Springer-Verlag, Vol. 1465, 1998, pp. 172-177.

[8] M. Malkin, T. Wu, and D. Boneh, "Experimenting with Shared Generation of RSA keys", In proceedings of the Internet Society's 1999 Symposium on Network and Distributed System Security (SNDSS), pp. 43–56.

[9] S. Micali, "Enhanced certificate revocation system", Technical memo, MIT/LCS/TM-542b, March 1996.

[10] M. Naor, K. Nissim, "Certificate revocation and certificate update", In proceedings of USENIX Security '98.

[11] M. Myers, R. Ankney, A. Malpani, S. Galperin and C. Adams, "X.509 Internet PKI Online Certificate Status Protocol - OCSP". IETF RFC 2560, June 1999.

[12] OpenSSL, `http://www.openssl.org`

[13] R. Rivest, "Can we eliminate Certificate Revocation Lists", Financial Cryptography – FC '98, Lecture Notes in Computer Science, Springer-Verlag, Vol. 1465, 1998, pp. 178-183.

[14] R. Rivest, A. Shamir and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", CACM, Vol. 21, No. 2, February 1978.

[15] SEM Eudora plug-in. `http://crypto.stanford.edu/semmail/`

Generating RSA Keys on a Handheld Using an Untrusted Server

Nagendra Modadugu
nagendra@cs.stanford.edu

Dan Boneh*
dabo@cs.stanford.edu

Michael Kim
mfk@cs.stanford.edu

Abstract

We show how to efficiently generate RSA keys on a low power handheld device with the help of an untrusted server. Most of the key generation work is offloaded onto the server. However, the server learns no information about the key it helped generate. We experiment with our techniques and show they result in up to a factor of 5 improvement in key generation time. The resulting RSA key looks like an RSA key for paranoids. It can be used for encryption and key exchange, but cannot be used for signatures.

1 Introduction

In recent years we have seen an explosion in the number of applications for handheld devices. Many of these applications require the ability to communicate securely with a remote device over an authenticated channel. Example applications include: (1) a wireless purchase using a cell phone, (2) remote secure synchronization with a PDA, (3) using a handheld device as an authentication token [2], and (4) handheld electronic wallets [3]. Many of these handheld applications require the ability to issue digital signatures on behalf of their users.

Currently, the RSA cryptosystem is the most widely used cryptosystem for key exchange and digital signatures: SSL commonly uses RSA-based key exchange, most PKI products use RSA certificates, etc. Unfortunately, RSA on a low power handheld device is somewhat problematic. For example, generating a 1024 bit RSA signature on the PalmPilot takes approximately 30 seconds. Nevertheless, since RSA is so commonly used on servers and desktops it is desirable to improve its performance on handhelds.

In this paper we consider the problem of *generating* RSA keys. Generating a 1024 bit RSA key on the PalmPilot can take as long as 15 minutes. The device locks up while generating the key and is inaccessible to the user. For wireless devices battery life time is a concern. Consider a user who is given a new cellphone application while traveling. The application may need to generate a key before it can function. Generating the key while the user is traveling will lock up the cellphone for some time and may completely drain the batteries.

The obvious solution is to allow the handheld to communicate with a desktop or server and have the server generate the key. The key can then be downloaded onto the handheld. The problem with this approach is that the server learns the user's private key. Consequently, the server must be trusted by the user. This approach limits mobility of the handheld application since users can only generate a key while communicating with their home domain. We would like to enable users to quickly generate an RSA key even when they cannot communicate with a trusted machine.

*Supported by NSF CCR-9732754.

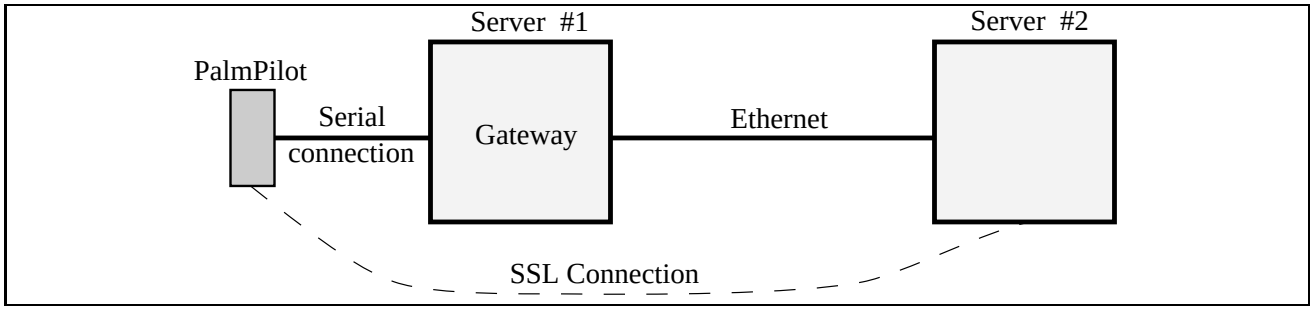

Figure 1: A two server configuration

We study the following question: can we speed up RSA key generation on a handheld with the help of an *untrusted server*? Our goal is to offload most of the key generation work onto the untrusted server. However, once the key is generated the server should have no information about the key it helped generate. This way the handheld can take advantage of the server's processing power without compromising the security of its keys.

Our best results show how to speed up the generation of *unbalanced* RSA keys. We describe these keys and explain how they are used in the next section. Our results come in two flavors. First, we show how to speed up key generation with the help of *two* untrusted servers. We assume that the two servers are unable to share information with each other. For instance, the two untrusted servers may be operated by different organizations. Using two untrusted servers we are able to speed up key generation by a factor of 5. We then show that a *single* untrusted server can be used to speed up key generation by a factor of 2. In Section 4 we discuss speeding up normal RSA key generation (as opposed to unbalanced keys).

We implemented and experimented with all our algorithms. We used the PalmPilot as an example handheld device since it is easy to program. Clearly our techniques apply to any low power handheld: pagers, cell phones, MP3 players, PDA's, etc. In our implementation, the PalmPilot connects to a desktop machine using the serial port. When a single server is used to help generate the key, the pilot communicates with the desktop using TCP/IP over the serial link. The desktop functions as the helping server. Note that there is no need to protect the serial connection. After all, since the desktop learns no information about the key it helped generate, an attacker snooping the connection will also learn nothing. When two servers are used, the desktop functions as a gateway enabling the pilot to communicate with the two servers. In this case, communication between the pilot and servers is protected by SSL to prevent eavesdropping by the gateway machine, and to prevent one server from listening in on communication intended for the other. Typically, the gateway machine functions as one of the two servers, as shown in Figure 1.

1.1 Timing cryptographic primitives on the PalmPilot

For completeness we list some running times for cryptographic operations on the PalmPilot. We used the Palm V which uses a 16.6MhZ Dragonball processor. Running times for DES, SHA-1, and RSA were obtained using a port of parts of SSLeay to the PalmPilot started by Ian Goldberg.

Algorithm	Time	Comment
DES encryption	4.9ms/block	
SHA-1	2.7ms/block	
1024 bit RSA key generation	15 minutes on average	
1024 bit RSA sig. generation	27.8 sec.	
1024 bit RSA sig. verify	0.758 sec.	$e = 3$
1024 bit RSA sig. verify	1.860 sec.	$e = 65537$

2 Preliminaries

2.1 Overview of RSA key generation

As a necessary background we give a brief overview of RSA key generation. Recall that an RSA key is made up of an n-bit modulus $N = pq$ and a pair of integers d, called the private exponent, and e, called the public exponent. Typically, N is the product of two large primes, each $n/2$ bits long. Throughout the paper we focus on generating a 1024 bit key (i.e. $n = 1024$). The algorithm to generate an RSA key is as follows:

Step 1: Repeat the following steps until two primes p, q are found:

- **a. Candidate** Pick a random 512 bit candidate value p.
- **b. Sieve** Using trial division, check that p is not divisible by any small primes (i.e. 2,3,5,7, etc.).
- **c. Test** Run a probabilistic primality test on the candidate. For simplicity one can view the test as checking that $g^{(p-1)/2} \equiv \pm 1 \pmod{p}$, where g is a random value in $1 \ldots p-1$. All primes will pass this test, while a composite will fail with overwhelming probability [10].

Step 2: Compute the product $N = pq$ (the product is 1024 bits long).

Step 3: Pick encryption and decryption exponents e and d where $e \cdot d = 1 \bmod \varphi(N)$ and $\varphi(N) = N - p - q + 1$.

The bulk of the key generation work takes place in step (1). Once the two primes p and q are found, steps (2) and (3) take negligible work. We note that trial division (step 1b) is frequently optimized by using a sieving algorithm. Sieving works as follows: once the candidate p is chosen in step (1a), the sieve is used to quickly find the closest integer to p that is not divisible by any small primes. The candidate p is then updated to be the integer found by the sieve. Throughout the paper we use a sieving algorithm attributed to Phil Zimmerman.

Our goal is to improve the performance of step (1). Within step (1), the exponentiation in step (1c) dominates the running time. Our goal is to offload the primality test to the server without exposing any information about the candidate being tested. Hence, the question is: how can we test that $g^{p-1} \bmod p = 1$ with the help of a server without revealing any information about p? To do so we must show how to carry out the exponentiation while solving two problems: (1) hiding the modulus p, and (2) hiding the exponent $p - 1$.

2.2 Unbalanced RSA keys

Our best results show how to speed up the generation of unbalanced RSA keys. An unbalanced key uses a modulus N of the form $N = p \cdot R$ where p is a 512 bit prime and R is a 4096 bit *random number*. One can show that with high probability R has a prime factor that is at least 512 bits long (the probability that it does not have such a factor is less than $1/2^{24}$). Consequently, the resulting modulus N is as hard to factor as a standard modulus $N = pq$.

An unbalanced key is used in the same way as standard RSA keys. The public key is $\langle e, N \rangle$ and the private key is $\langle d, N \rangle$. We require that $e \cdot d = 1 \bmod p - 1$. Suppose p is m-bits long. The system can be used to encrypt messages shorter than m bits. As in standard RSA, to encrypt a message M, whose length is much shorter than m bits, the sender first applies a randomized padding mechanism, such as OAEP [4, 9]. The padding mechanism results in an $m-1$ bit integer P (note that $P < p$). The sender then constructs the ciphertext by computing $C = P^e \bmod N$. Note that the ciphertext is as big as N. To decrypt a ciphertext C, the receiver first computes $C_p = C \bmod p$ and then recovers P by computing $P = C_p^d \bmod p$. The plaintext M is then easily extracted from P. Since decryption is done modulo p it is as fast as standard RSA.

The technique described above for using an unbalanced key is similar to Shamir's "RSA for paranoids" [11]. It shows that unbalanced keys can be used for encryption/decryption and key exchange. Unfortunately, unbalanced keys cannot be used for digital signatures. We note that some attacks against RSA for paranoids have been recently proposed [5]. However, these attacks do not apply when one uses proper padding prior to encryption. In particular, when OAEP padding is used [4] the attacks cannot succeed since the security of OAEP (in the random oracle model) only relies on the fact that the function $f : \{0, \ldots, 2^{m-1}\} \to \mathbb{Z}_N$ defined by $f(x) = x^e \bmod N$ is a one-to-one trapdoor one way function.

3 Generating an unbalanced RSA key with the help of untrusted servers

We show how RSA key generation can be significantly sped up by allowing the PalmPilot to interact with untrusted servers. At the end of the computation the servers should know nothing about the key they helped generate. We begin by showing how *two* untrusted servers can help the Pilot generate RSA keys. The assumption is that these two servers cannot exchange information with each other. To ensure that an attacker cannot eavesdrop on the network and obtain the information being sent to both servers, our full implementation protects the connection between the Pilot and the servers using SSL. Typically, the machine to which the pilot is connected can be used as one of the untrusted servers (Figure 1). We then show how to speed up key generation with the help of a *single* server. In this case there is no need to protect the connection.

3.1 Generating keys with the help of two servers

Our goal is to generate a modulus of the form $N = pR$ where p is a 512-bit prime and R is a 4096-bit random number. To offload the primality test onto the servers we must hide the modulus p and the exponent $p - 1$. To hide the modulus p we intend to multiply it by a random number R and send the resulting $N = pR$ to the servers. The server will perform computations modulo $N = pR$. If it turns out that p is prime, then sending N to the servers does not expose any information about p or R. If

p is not prime we start over. To hide the exponent $p-1$ used in the primality test we intend to share it among the two servers. Individually, neither one of the servers will learn any information.

Our algorithm for generating an unbalanced RSA modulus $N = pR$ is as follows. The algorithm repeats the following steps until an unbalanced key is generated:

Step 1: Pilot generates a 512 bit candidate p that is not divisible by small primes and a 4096 bit random number R. We require that $p = 3 \bmod 4$.

Step 2: Pilot computes $N = p \cdot R$.

Step 3: Pilot picks random integers s_1 and s_2 in the range $[-p, p]$ such that $s_1 + s_2 = (p-1)/2$. It also picks a random $g \in \mathbb{Z}_N^*$.

Step 4: Pilot sends $\langle N, g, s_1 \rangle$ to server 1 and $\langle N, g, -s_2 \rangle$ to server 2.

Step 5: Server 1 computes $X_1 = g^{s_1} \bmod N$. Server 2 computes $X_2 = g^{(-s_2)} \bmod N$. Both results X_1 and X_2 are sent back to the pilot.

Step 6: Pilot checks whether $X_1 = \pm X_2 \bmod p$. If equality holds, then $N = pR$ is declared as a potential unbalanced RSA modulus. Otherwise, the algorithm is restarted in Step 1.

Step 7: The Pilot locally runs a probabilistic primality test to verify that p is prime. This is done to ensure that the servers returned correct values.

First, we verify the soundness of the algorithm. In step 6 the Pilot verifies that $g^{s_1} \cdot g^{s_2} = g^{(p-1)/2} = \pm 1 \bmod p$. If the test is satisfied then p is very likely to be prime. Then step 7 ensures that p is in fact prime and that the servers did not respond incorrectly. When generating a 1024 bit RSA key, a single primality test takes little time compared to the search for a 512 bit prime. Hence, Step 7 adds very little to the total running time.

During the search for the prime p, the only computation carried out by the pilot is the probable prime generation and the computation of s_1 and s_2. The time to construct s_1 and s_2 is negligible. On the other hand, generating the probable prime p requires a sieve to ensure that p is not divisible by small factors. As we shall see in Section 5 the sieve is the bottleneck. This is unusual since in standard RSA modulus generation sieving takes only a small fraction of the entire computation. We use a sieving method attributed to Phil Zimmerman. We note that faster sieves exist, but they result in an insecurity of our algorithm.

Security To analyze the security properties of the algorithm we must argue that the untrusted servers learn no information of value to them. During the search for the RSA modulus many candidates are generated. Since these candidates are independent of each other, any information the servers learn about rejected candidates does not help them in attacking the final chosen RSA modulus. Once the final modulus $N = pR$ is generated in Step 2, each server is sent the value of N and s_i where i is either 1 or 2. The modulus N will become public anyhow (it is part of the public key) and hence reveals no new information. Now, assuming servers 1 and 2 cannot communicate, the value s_1 is simply a random number (from Server 1's point of view). Server 1 could have just as easily picked a random number in the range $[-N, N]$ itself. Hence, s_1 reveals no new information to Server 1 (formally, a simulation argument shows that s_1 reveals at most two bits). The same holds for Server 2. Hence, as long as Server 1 and Server 2 cannot communicate, no useful information is revealed about the factorization of N. We note that if the servers are able to communicate, they can factor N.

Performance The number of iterations until a modulus is found is identical to local generation of an (unbalanced) modulus on the PalmPilot. However, each iteration is much faster than the classic RSA key generation approach of Section 2.1. After all, we offloaded the expensive exponentiation to a fast Pentium machine. As we shall see in Section 5.2, the total running time is reduced by a factor of 5.

3.2 Generating keys with the help of a single server

Next, we show how a *single* untrusted server can be used to reduce the time to generate an RSA key on the PalmPilot. Once the key is generated, the server has no information regarding the key it helped generate. Typically, the pilot connects to the helping server directly through the serial or infrared ports.

As before we need to compute $g^{(p-1)/2} \bmod p$ to test whether p is prime. Our technique involves reducing the size of the exponent using the help of the server and hence speeding up exponentiation on the pilot. The algorithm repeats the following steps until an unbalanced modulus is found:

Step 1: Pilot generates a 512 bit candidate p that is not divisible by small primes and a 4096 bit random number R. We require that $p = 3 \bmod 4$.

Step 2: Pilot computes $N = p \cdot R$. It picks a random $g \in \mathbb{Z}_N^*$.

Step 3: Pilot picks a random 160 bit integer r and a random 512 bit integer a. It computes $z = r + a(p-1)/2$.

Step 4: Pilot sends $\langle N, g, z \rangle$ to the server.

Step 5: The server computes $X = g^z \bmod N$ and sends X back to the Pilot.

Step 6: Pilot computes $Y = g^r \bmod p$.

Step 7: Pilot checks if $X = \pm Y \bmod p$. If so then the algorithm is finished and $N = pR$ is declared as a potential unbalanced RSA modulus. Otherwise, the algorithm is restarted in Step 1.

Step 8: The Pilot locally runs a probabilistic primality test to verify that p is prime.

To verify soundness observe that N will make it to step 8 only if $X = \pm Y$ i.e. $g^{r+a(p-1)/2} = \pm g^r \bmod p$. As before, this condition is always satisfied if p is prime. The test will fail with overwhelming probability if p is not prime. Hence, once step 8 is reached the modulus $N = pR$ is very likely to be an unbalanced modulus. The test is Step 8 takes little time compared to the entire search for the 512-bit prime.

Performance Since we are generating an unbalanced modulus the number of iterations until N is found is the same as in local generation of such a modulus on the PalmPilot. Within each iteration the Pilot generates p and R using a sieve and then computes $Y = g^r \bmod p$ (in step 6). However, r is only 160 bits long. This is much shorter than when a key is generated without the help of a server. In that case the Pilot has to compute an exponentiation where the exponent is 512 bits long. Hence, we reduce the exponentiation time by approximately a factor of three. Total key generation time is reduced by a factor of 2, due to the overhead of sieving.

Recall that in Step 6 the Pilot computes $Y = g^r \bmod p$ where r is a 160-bit integer. This step can be further sped up with the help of the server. Let $A = 2^{40}$ and write $r = r_0 + r_1 A + r_2 A^2 + r_3 A^3$ where r_0, r_1, r_2, r_3 are all in the range $[0, A]$. In Step 5 the server could send back the vector $\vec{R} = \langle g^A, g^{A^2}, g^{A^3} \rangle \bmod N$ in addition to sending X. Let $\vec{R} = \langle R_1, R_2, R_3 \rangle$. Then in Step 6 the Pilot only has to compute $Y = g^{r_0} \cdot R_1^{r_1} \cdot R_2^{r_2} \cdot R_3^{r_3} \bmod p$. Using Simultaneous Multiple Exponentiation [7, p. 617] Step 6 can now be done in approximately half the time of computing $Y = g^r \bmod p$ on the Pilot directly. This improvement reduces the total exponentiation time on the Pilot by an additional factor of 2 .

Security In the last iteration, when the final p and R are chosen, the server learns the value $z = a(p-1) + r$. Although z is a "random looking" 1024 bit number, it does contain some information about p. In particular, $z \bmod p-1$ is very small (only 160 bits long). The question is whether z helps an adversary break the resulting key. The best known algorithm for doing so requires $2^{r/2}$ modular exponentiations. Due to our choice of 160 bits for r, the algorithm has security of approximately 2^{80}. This is good enough since a 1024 bits RSA key offers security of 2^{80} anyhow. Nevertheless, the security of the scheme is heuristic since it depends on the assumption that no faster algorithm exists for factoring N given z. More precisely, the scheme depends on the following "$(p-1)$-multiple assumption":

$(p-1)$-*multiple assumption:* Let A_n be the set of integers $N = pq$ where p and q are both n-bit primes. Let m be an integer so that the fastest algorithm for factoring a random element $N \in A_n$ runs in time at least $2^{m/2}$. Then the two distributions: $\langle N, r + a(p-1)/2 \rangle$ and $\langle N, x \rangle$ cannot be distinguished with non-negligible probability by an algorithm whose running time is less than $2^{m/2}$. Here N is randomly chosen in A_n, a is randomly chosen in $[0, p]$, r is randomly chosen in $[0, 2^m]$, and x is randomly chosen in $[0, p^2/2]$.

Based on the $(p-1)$-multiple assumption, the integer z given to the server contains no more statistical information than a random number in the range $[0, p^2]$. Hence, the server learns no new useful information from z.

As before, since the generated key is an unbalanced key it can only be used for encryption/decryption and key exchange. It cannot be used for signatures.

4 Generating standard RSA keys

One could wonder whether the techniques described in the previous sections can be used to speed up generation of standard RSA keys. We show that at the moment these techniques do not appear to improve the generation time for a 1024 bit key. For shorter keys, e.g. 512 bits keys, we get a small improvement. In what follows we show how to generate a normal RSA key, $N = pq$, with the help of two servers.

We wish to generate an RSA modulus $N = pq$ where p and q are each 512-bits long. As before, we wish to offload the primality test to the servers. To do so we must hide the moduli p and q and the exponents $p-1$ and $q-1$. The basic idea is to *simultaneously* test primality of both p and q. For each pair of candidates p and q the Pilot computes $N = pq$ and sends N to the servers. The servers carry out the exponentiations modulo N. To hide the exponents $p-1$ and $q-1$ we share them among the two servers as in the last section.

The resulting algorithm is similar to that for generating unbalanced keys. In fact, the server-side is

identical. The algorithm works as follows. Repeat the following steps until a standard RSA modulus is found:

Step 1: Pilot generates two candidates p, q so that neither one is divisible by small primes. We refer to p and q as probable primes.

Step 2: Pilot computes $N = p \cdot q$ and $\varphi(N) = N - p - q + 1$. Pilot picks a random $g \in \mathbb{Z}_N^*$.

Step 3: Pilot picks random integers φ_1 and φ_2 in the range $[-N, N]$ such that $\varphi_1 + \varphi_2 = \varphi(N)/4$.

Step 4: Pilot sends $\langle N, g, \varphi_1 \rangle$ to server 1 and $\langle N, g, -\varphi_2 \rangle$ to server 2.

Step 5: Server 1 computes $X_1 = g^{\varphi_1} \pmod{N}$. Server 2 computers $X_2 = g^{-\varphi_2} \pmod{N}$. Both results X_1 and X_2 are sent back to the pilot.

Step 6: Pilot checks if $X_1 = \pm X_2 \bmod N$. If so, the algorithm is finished and $N = pq$ is declared as a potential RSA modulus. Otherwise, the algorithm is restarted in Step 1.

Step 7: The Pilot locally runs a probabilistic primality test to verify that p and q are prime. This is done to ensure that the servers returned correct values.

First, we verify soundness of the algorithm. In step 6 the Pilot is testing that $X_1 = \pm X_2$, namely that $g^{\varphi_1} = g^{-\varphi_2} \bmod N$. That is, we check that $g^{\varphi_1 + \varphi_2} = g^{\varphi(N)/4} = \pm 1 \bmod N$. Clearly, this condition holds if p and q are both primes. Furthermore, it will fail with overwhelming probability if either p or q are not prime. Hence, Step 7 is reached only if $N = pq$ is extremely likely to be a proper RSA modulus. Step 7 then locally ensures that p and q are primes.

Security To analyze the security properties of the algorithm we must argue that the untrusted servers learn no information of value to them. During the search for the RSA modulus many candidates are generated. Since these candidates are independent of each other, any information the servers learn about rejected candidates does not help them in attacking the final chosen RSA modulus. Once the final modulus $N = pq$ is generated in Step 2, each server is sent the value of N and φ_i where i is either 1 or 2. The modulus N will become public anyhow (it is part of the public key) and hence reveals no new information. Now, assuming servers 1 and 2 cannot communicate, the value φ_1 is simply a random number (from Server 1's point of view). Server 1 could have just as easily picked a random number in the range $[-N, N]$ itself. Hence, φ_1 reveals no new information to Server 1. As long as Server 1 and Server 2 cannot communicate, no useful information is revealed about the factorization of N. If the servers are able to communicate, they can factor N.

Performance Each iteration in our algorithm is much faster than the classic RSA key generation approach of Section 2.1 — we offloaded the expensive exponentiation to a fast Pentium machine. Unfortunately, the number of iterations required until an RSA modulus is found is higher. More precisely, suppose in the classic approach one requires k iterations on average until a 512-bit prime is found. Then the total number of iterations to find two primes is $2k$ on average. In contrast, in our approach both p and q must be simultaneously prime. Hence, k^2 iterations are required. We refer to this effect as a *quadratic slowdown*. When generating a 1024 bit modulus the value of k is approximately 14. So even though we are able to speed up each iteration by a factor of 5, there are seven times as many iterations on average. Therefore when generating a standard 1024 bit key these techniques do not improve the running time. When generating a shorter key, e.g. a 512 bit key, the

quadratic slowdown penalty is less severe since k is smaller (9 rather than 14). For such short keys we obtain a small improvement in performance.

Similarly, when generating keys with the help of a *single* server, the quadratic slowdown outweighs the reduction in time per iteration. It is an open problem to speed up server aided generation of standard RSA keys.

5 Experiments and implementation details

5.1 Implementation details

The two main components of our implementation were the cryptographic and networking modules. SSLeay provided for the cryptographic code on both the server (Pentium II 400Mhz) and PalmPilot side. In the case of the Pilot, we used SSLeay code that had been previously ported by Ian Goldberg.

5.1.1 Networking

We connect the pilot to a Windows NT gateway running RAS (Remote Access Service) through a serial-to-serial interface. The function of the gateway was to provide TCP/IP access to the network.

In our single server implementation, we used the gateway as our assisting server while in our dual server implementation, we used the gateway and another local machine.

Our networking layer abstracts the secure communication of BigIntegers to and from the PalmPilot. The network layer packs a number of BigIntegers into a buffer and sends the entire buffer at once. The receiving side unpacks the buffer and processes it as required.

5.2 Experiments

Tables 1 and 2 show the results we obtained when generating 512, 768 and 1024 bit RSA keys. The network traffic column measures the amount of data (in bytes) exchanged between the Pilot and the servers. We generate keys using three methods:

(1) Local: Key generated locally on the Pilot (no interaction with a server).

(2) One server: Pilot aided by a single untrusted server.

(3) Two servers: Pilot aided by two untrusted servers.

As expected we see that generating unbalanced keys with the aid of one or two servers leads to a performance improvement over generating keys locally on the PalmPilot. The rest of this section discusses these experimental results.

We note that the key factor that determines the time it takes to generate an RSA key is the time *per iteration* (the time to sieve and exponentiate *one* probable prime p). This number is more meaningful than the total running since since the total time has a high variance. More precisely, the number of iterations until a prime is found has high variance. Our tables state the average number of iterations we obtained.

In our experiments, we carried out trial division on a candidate prime using the first 2048 primes (upto approximately 17,000). In all our experiments we observed that the server's responses are

		Sieve time(ms)	Server time(ms)	Exp. time(ms)	total time/ iter.(ms)	average num. iter.	total time	net traf.
Local	unbal.	3,805		21,233	25,038	18.16	7.5min.	
Local	norm.	3,805		21,233	25,038	36.32	15.16min.	
One serv.	unbal.	3,516	955	6,995	11,467	14.5	2.7min.	5,568
Two serv.	unbal.	3,587	1,462		5,156	12.75	1.1min	8,160
Two serv.	norm.	7,850	820	0	8,720	406	59min	311,808

Table 1: Statistics for different key generation methods (1024 bit keys)

instantaneous compared to the Pilot's processing time. Consequently, improving server performance will only marginally affect the overall running time.

5.2.1 Generating a 1024 bit key

Table 1 shows detailed timing measurements for generating 1024 bit RSA keys. Our breakdown of timing measurements follows the description in Section 2.1. The first column shows the time to pick a probable prime, the second shows the time the Pilot spent waiting for the server to respond, the third shows the time to exponentiate on the PalmPilot (not used in the two-server mode). The last column shows the total network traffic (in bytes).

The first two rows in Table 1 measure the time to generate keys on the Pilot. The first column represents the time to generate an unbalanced key, the second represents the time to generate a normal $N = pq$ key. Since an unbalanced key requires only one prime (the other is a random number) the number of iterations for locally generating an unbalanced key is half that for generating a normal key.

When comparing the time per iteration for local generation and two server generation, we see that using two servers we get an improvement of a factor of 5. Using one server we obtain an improvement of a factor of 2. The average number of iterations is approximately the same in all three methods. Note that the improvements are a result of speeding up (or eliminating) the exponentiation step on the PalmPilot. Observe that when two servers are used the bottleneck is the sieving time — the time to generate a probable prime p.

On average, 406 iterations are needed to generate a normal RSA key ($N = pq$) with the aid of two servers. The large number of iterations is a result of the quadratic slowdown discussed in Section 4. Even though each iteration is much faster than the corresponding value for local generation, we end up hurting the total generation time.

Our algorithms require only a few kilobytes of data transfer between the Pilot and the servers. The traffic generated is linear in the number of iterations which explains the large figure for two server normal key generation.

5.2.2 Generating various key sizes

From Table 2 we see that the total iteration time increases almost linearly with key size for dual server aided generation. Indeed, the dominant component of each iteration is sieving, which takes linear time as a function of the key size. The expected total time for generating the key is the product of the time-per-iteration and the expected-number-of-iterations.

Observe that the improvement over local generation is less significant for shorter keys than for

	512 bits			768 bits			1024 bits		
	num. iter.	time/ iter.(ms)	net traf.	num. iter.	time/ iter.(ms)	net traf.	num. iter.	time/ iter.(ms)	net traf.
Local unbal.	9.15	3,550		10.53	8,215		18.16	25,038	
Local norm.	18.3	3,550		21.1	8,215		36.32	25,038	
One serv. norm.	9.3	4,546	1,785	14.8	7,644	4,262	14.5	11,467	5,568
Two serv. unbal.	9	2,492	2,880	12.55	3,687	6,024	12.75	5,156	8,160
Two serv. norm.	26	4,364	9,984	119.7	6,560	68,947	406	8,720	311808

Table 2: Statistics for different key sizes

longer keys. For instance, for a 512 bit key, two servers improve performance by only 50%. For a 1024 bit key the improvement is a factor of 5. The reason is that for smaller keys, the primality test is less of a dominating factor in the running time per iteration (we use the same size sieve, 2048 primes, for all key sizes). Hence, reducing the exponentiation time has less of an effect on the the total time per iteration.

6 Conclusions

At the present using RSA on a low power handheld is problematic. In this paper we study whether RSA's performance can be improved without a loss of security. In particular, we ask whether an untrusted server can aid in RSA key generation. We wish to offload most of the work to the server without leaking any of the handheld's secrets.

We showed a significant improvement in the time it takes to generate an *unbalanced* RSA key. With the help of two isolated servers we obtained a speed up of a factor of 5. With the help of a *single* server we obtained a speed up of a factor of 2. For normal RSA keys, $N = pq$, we cannot improve the running time due do the *quadratic slowdown* problem discussed in Section 4. It is an open problem to speed up the generation of a normal RSA key using a single server. In all our algorithms the load on the server is minimal; our experiments show that even though the server is doing most of the work, the PalmPilot does not produce candidates fast enough to fully occupy the server. Our code available for anyone wishing to experiment with it.

Acknowledgments

We thank Certicom for providing us with SSL libraries for the PalmPilot.

References

[1] N. Asokan, G. Tsudik and M. Waidner, "Server-Supported Signatures", Journal of Computer Security, Vol. 5, No. 1, pp. 91–108, 1997.

[2] D. Balfanz, E. Felten, "Hand-Held Computers Can Be Better Smart Cards", to appear in the 8th USENIX Security Symposium.

[3] D. Boneh, N. Daswani, "Experimenting with electronic commerce on the PalmPilot", in proc. of Financial-Crypto '99, Lecture Notes in Computer Science, Vol. 1648, Springer-Verlag, pp. 1–16, 1999.

[4] M. Bellare, P. Rogaway, "Optimal asymmetric encryption – How to encrypt with RSA", in proc. Eurocrypt '94.

[5] H. Gilbert, D. Gupta, A. M. Odlyzko, and J.-J. Quisquater, "Attacks on Shamir's 'RSA for paranoids," Information Processing Letters 68 (1998), pp. 197–199.

[6] T. Matsumoto, K. Kato, H. Imai, "Speeding up secret computations with insecure auxiliary devices", In proc. of Crypto '88, Lecture Notes in Computer Science, Vol. 403, Springer-Verlag, pp. 497–506, 1998.

[7] A. Menezes, P. van Oorschot and S. Vanstone, "Handbook of Applied Cryptography", CRC Press, 1996.

[8] P. Nguyen, J. Stern, "The Beguin-Quisquater Server-Aided RSA Protocol from Crypto'95 is not secure", in proc. of AsiaCrypt '98, Lecture Notes in Computer Science, Vol. 1514, Springer-Verlag, pp. 372–380, 1998.

[9] Public Key Cryptography Standards (PKCS), No. 1, "RSA Encryption standard", `http://www.rsa.com/rsalabs/pubs/PKCS/`.

[10] R. Rivest, "Finding four million large random primes", In proc. of Crypto '90, Lecture Notes in Computer Science, Vol. 537, Springer-Verlag, pp. 625–626, 1997.

[11] A. Shamir, "RSA for paranoids", CryptoBytes, Vol. 1, No. 3, 1995.

Server-Supported Signatures *

N. Asokan†
IBM Research Division, Zurich Research Laboratory, Switzerland and
University of Waterloo, Canada

G. Tsudik‡
University of Southern California,
Information Sciences Institute, USA

M. Waidner§
IBM Research Division, Zurich Research Laboratory, Switzerland

Journal of Computer Security, to appear in Fall 1997

Abstract

Non-repudiation is one of the most important security services. In this paper we present a novel non-repudiation technique, called **Server-Supported Signatures, S^3.** It is based on *one-way hash functions* and *traditional digital signatures.* One of its highlights is that for ordinary users the use of asymmetric cryptography is limited to signature *verification.* S^3 is efficient in terms of computational, communication and storage costs. It also offers a degree of security comparable to that of existing techniques based on asymmetric cryptography.

Keywords: digital signatures, non-repudiation, electronic commerce, network security, distributed systems, mobility.

*A previous version of this work was presented at ESORICS '96 [1].

†e-mail: aso@zurich.ibm.com
‡e-mail: gts@isi.edu
§e-mail: wmi@zurich.ibm.com

1 Introduction

Computers and communication networks have become an integral part of many people's daily lives. Systems to facilitate commercial and other transactions have been built on top of large open computer networks. These transactions must often have some legal significance if they are to be useful in real life. Non-repudiation is one of the essential services necessary for attaching legal significance to transactions and information transfer in general.

Existing techniques for non-repudiation are based primarily on either symmetric or asymmetric cryptography. Practically secure symmetric techniques are computationally more efficient but require unconditional trust in third parties. "Unconditional" means that if such a third party cheats, the victim cannot prove this to an arbiter (e.g., a court). Practically secure asymmetric techniques (which we refer to as "traditional digital signatures") are computationally less efficient but can be constructed in a way that allows one to prove cheating by any third parties involved. We call a third party whose cheating can be proven to an arbiter a **verifiable third party.**

We present a novel non-repudiation technique called **Server-Supported Signatures, S^3.** It is based on *one-way hash functions* and *traditional digital signatures.* Like well-constructed asymmetric techniques, S^3 uses only verifiable third parties. However, for ordinary users, S^3 limits the use of asymmetric cryptographic techniques to signature *verification.* All signature *generations* are done by third parties, called *signature servers.* For some signature schemes, e.g., RSA with a public exponent of 3, verifying signatures is significantly more efficient than generating them.

2 Background and Motivation

The International Standardization Organization (ISO) is in the process of standardizing techniques to provide non-repudiation services in open networks. Current versions of the draft ISO standards [5] identify various classes of non-repudiation services. Two of these are of particular interest:

- **Non-repudiation of Origin (NRO)** guarantees that the originator of a message cannot later deny having originated that message.
- **Non-repudiation of Receipt (NRR)**[1] guarantees that the recipient of a message cannot deny having received that message.

Non-repudiation for a particular message is obtained by constructing a **non-repudiation token.** The non-repudiation token must be such that it can be verified by:

- the intended recipients of the token (e.g., in the case of NRO, the recipient of the message; in the case of NRR, the originator of the message), and
- in case of a dispute, by a mutually acceptable *arbiter.*

The draft ISO standards divide non-repudiation techniques into two classes:

- *Asymmetric non-repudiation techniques* are based on digital signature schemes using public-key cryptography. The main (and probably the only) difficulty in using digital signature schemes is the computational cost involved. This is a particularly serious issue when "anemic" portable devices (like mobile phones) are involved.

  Non-repudiation is based on certification of the signer's public key by a *certification authority.* Trust in this certification authority can be minimized by an appropriate *registration procedure.* For example, the signer and the authority may be required to sign a paper contract listing the signer's and certification authority's public keys, responsibilities, and liabilities, possibly in front of a notary public. In the worst case, the certification authority could cheat the user by issuing a certificate with a public key

[1] ISO documents call this "non-repudiation of delivery (NRD)." We use the term "receipt" because we feel that the term "delivery" is more appropriate to describe the function performed by the message transport system.

chosen by a cheater. But the supposed signer could deny all signatures based on this forged certificate by citing the contract signed during registration. Thus, trust is reduced to trust in the verifiability of the registration procedure.

- *Symmetric non-repudiation techniques* are based on symmetric message authentication codes (MACs) and trusted third parties that act as witnesses. Generating and verifying message authentication codes are typically low-cost operations compared to digital signature operations.

  The signer has to trust the third party unconditionally, which means that the third party could cheat the user without giving the user any chance to deny forged messages. One could reduce this trust by using several third parties in parallel or by putting the third party into tamper resistant hardware. These two approaches increase both cost and complexity but neither of them solves the problem completely.

In the following we present a new, low-cost technique for non-repudiation services, called *server-supported signatures*. It uses both traditional digital signatures (based on asymmetric cryptographic techniques) and one-way hash functions in order to minimize the computational costs for ordinary users. Our main motivation arises from the typical mobile computing environments where the mobile entities have considerably less computing power than do static entities.

3 Server-Supported Signatures for Non-repudiation of Origin

3.1 Preliminaries: One-way Hash Functions

Intuitively, a one-way function $f()$ is a function such that given an input string x it is easy to compute $f(x)$, but given a randomly chosen y it is computationally infeasible to find an x' such that $f(x') = y$. A one-way hash function is a one-way function $h()$ that operates on arbitrary-length inputs to produce a fixed length value. The term x is called a *pre-image* of $h(x)$. A one-way hash function $h()$ is said to be *collision-resistant* if it is computationally infeasible to find any two strings x and x' such that $h(x) = h(x')$. Collision-resistance implies one-wayness [13, Section 7.2]. A number of efficient and allegedly one-way hash functions, such as SHA[8], have been invented. One-way hash functions can be recursively applied to an input string. The notation $h^i(x)$ denotes the result of applying $h()$ i times recursively to an input x. That is,

$$h^i(x) = \underbrace{h(h(h(\ldots h}_{i \text{ times}}(x)\ldots)))$$

Such recursive application results in a *hash-chain* that is generated from the original input string:

$$h^0(x) = x, h^1(x), \ldots, h^n(x)$$

3.2 Model and Notation

We distinguish three types of entities in the system:

- *Users* – participants in the system who wish to avail themselves of the non-repudiation service while sending and receiving messages among themselves.
- *Signature Servers* – special entities responsible for actually generating the non-repudiation tokens on behalf of the users.
- *Certification Authorities* – special entities responsible for linking public keys with identities of users and servers.

Signature servers and certification authorities will be verifiable third parties from the users' point of view.

All entities agree on a one-way, collision-resistant hash-function $h()$ and a digital signature scheme. Entities should "personalise" the hash function. For example, this can be done by always including their unique name as an argument: using $h(O, x)$, where O is the entity computing the one-way hash. We use $h_O()$ to refer to the personalised hash function used by O.

The result of digitally signing a message x with signature key SK is denoted by $(x)SK$. We also assume that, given $(x)SK$, anyone can extract x from it provided they have the public key corresponding to SK. The users' security depends on the one-way property of $h_O()$, which must hold even against the servers. In practice, this is not a problem because hash functions such as SHA are one-way for *all* parties. We note, however, that so-called cryptographically strong hash-functions are usually invertible for the party that generated the hash-function.

In order to minimize the computational overhead for users, $h_O()$ must be efficiently computable, and digital signatures must be efficiently *verifiable*. Only signature servers and certification authorities need to have the ability to *generate* signatures. SHA as hash function and RSA with public exponent 3 as signature scheme would be reasonable choices.

Each user, O (O as in *O*riginator) generates a secret key, K_O, randomly chosen from the range of $h_O()$. Based on K_O, user O computes the hash chain $K_O^0, K_O^1, \ldots K_O^n$, where

$$K_O^0 = K_O,\ K_O^i = h_O^i(K_O) = h_O(K_O^{i-1})$$

$PK_O = K_O^n$ constitutes O's *root public key*. Root public key K_O^n will enable O to authenticate n messages. This is not a limitation: before the old root public key is consumed completely a new root public key can be generated and authenticated using the old root public key.

Each signature server S generates a pair of secret and public keys (SK_S, PK_S) of the digital signature scheme. Each certification authority, CA, does the same. The CA is responsible for verifiably binding a user O (server S) to her root public key PK_O (its public key PK_S). We assume that the registration procedure is constructed such that CA becomes a verifiable third party.

Notation Summary

$h_O()$ - one-way collision-resistant hash function for O
SK_X - secret key known only to entity X
K_O^i - user O's $(n-i)$-th public key
$(x)SK$ - digital signature on message x with secret key SK
$[m]$ - Message m sent via a confidential channel

3.3 Initialization

To participate in the system, a user O chooses a signature server S that shall be responsible for generating signatures on O's behalf, generates a random secret key K_O, and constructs the hash chain. As will be described below, O can cause S to transfer the signature generation responsibility to another signature server S', if required (e.g., because O is a mobile user who wishes to always use the closest server available).

O submits the root public key $PK_O = K_O^n$ to a CA for certification. A certificate for O's root public key is of the form

$$Cert_O = (O, n, PK_O, S)SK_{CA}$$

We ignore all information typically contained in a certificate but not relevant to the discussion at hand, e.g. organizational data such as serial numbers and expiration dates. The registration performed by O and CA must be verifiable, as discussed above. CA may make the certificate available to anyone via a directory service. O then deposits the certificate received from CA with S.

Each signature server S acquires a certificate containing PK_S from its CA. As these are ordinary public key certificates, we do not describe them here.

For the sake of simplicity, we do not include the certificates in the following protocols. They might be attached to other messages or retrieved using a directory service. We assume that the necessary certificates are always available to anyone who needs to verify a signature.

3.4 Generating NRO Tokens

The basic idea is to exploit the digital signature generation capability of a signature server to provide non-repudiation services to ordinary users. The basic protocol, providing non-repudiation of origin, is illustrated

in Figure 1. We assume that a user O wants to send a message m along with an NRO token to some recipient R. The first protocol run uses $i = n$; i is decreased during each run.

1. O begins by sending (O, m, i) to its signature server S along with O's current public key K_O^i in the first protocol flow (in case O does not want to reveal the message to S for privacy reasons, m can be replaced by a randomised hash of m, computed using a collision-resistant hash function).

2. S verifies the received public key based on O's root public key (and O's certificate obtained from CA), i.e., checks that $h_O^{n-i}(K_O^i) = PK_O$. The signature server S has to ensure that only one NRO can be created for a given (O, i, K_O^i). If a message on behalf of O containing K_O^i has not yet been signed, S signs (O, m, i, K_O^i), records K_O^i as consumed, and sends the signature (which we call the *candidate non-repudiation token*) back to O in the second flow.

3. O verifies the received signature and records K_O^i as consumed by replacing i by $i-1$. The NRO token for R now consists of

$$(O, m, i, K_O^i)SK_S, K_O^{i-1}$$

   O produces this token which actually authenticates m, by revealing K_O^{i-1}.

   In Figure 1 we assumed that the NRO token is sent to R via S in the third flow. Alternatively, O can send the token directly to R.

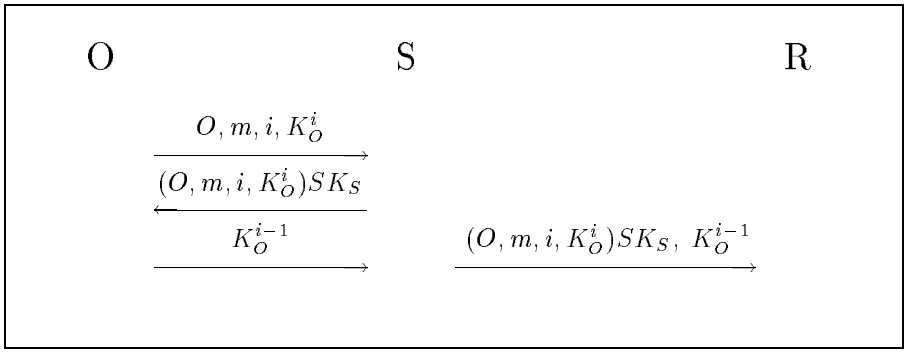

Figure 1: Protocol providing non-repudiation of origin.

K_O^i is referred as the **token public key** of the $(n-i+1)$th non-repudiation token, $(O, m, i, K_O^i)SK_S, K_O^{i-1}$. Note that O must consume the token public keys in sequence and must not skip any of them. In particular, O must not ask for a signature using K_O^{i-1} as token public key unless she has received S's signature under K_O^i. Otherwise, S could use that to create a fake non-repudiation token, which O cannot repudiate during a dispute.

3.5 Dispute Resolution

In case of a dispute, R can submit the NRO to an arbiter. The arbiter will verify the following:

- the public keys are certified by CA,
- the signature in the token by the signature server is valid,
- the token public key is in fact a hash of the alleged pre-image in the token, and
- the root public key can be derived from the token public key by repeated hashing.

If these checks are successful, then the originator is allowed the opportunity to *repudiate* the token by

- proving that CA cheated:
  - If O has registered with CA, O can show a certificate on a different root public key.
  - Otherwise CA will be asked to prove that the root public key was registered by O (i.e., by showing the signed contract with O).
- proving that S cheated by showing a different non-repudiation token corresponding to the same token public key.

Note that in case CA is honest, to claim falsely that O has sent a message m', a cheating R has to produce an NRO token of the form:

$$(O, m', i, K_O^i)SK_S, K_O^{i-1}$$

If O has not revealed K_O^{i-1} yet, then one-wayness of $h_O()$ implies that anyone else will find it computationally infeasible to generate this NRO token, even if K_O^i is known. If O has already revealed K_O^{i-1} she must have sent K_O^i to S before. According to the protocol, O reveals K_O^{i-1} only if she has received a signature from S under K_O^i which satisfied her. Therefore, O can show a different token corresponding to the same token public key.

Suppose an adversary of O successfully breaks the one-wayness of $h_O()$ and obtains an NRO token of the form:

$$(O, m', i, K_O^i)SK_S, x$$

where $h_O(x) = h_O(K_O^{i-1})$. If x is different from K_O^{i-1}, then on being challenged with this NRO token, O can reveal K_O^{i-1}, proving that the system has been broken. This is known as the "fail-stop" property. Assuming that h_O has a uniform distribution, the domain used must be larger than the range of h_O in order to achieve a reasonable level of fail-stop property. We can do this by slightly modifying the building procedure to include a random padding to the input of h_O during the computation of every link in the chain. The sequence of random pads used are generated using a pseudo-random number generator whose seed is committed to in the certificate.

4 Server-Supported Signatures for Non-repudiation of Origin and Receipt

Non-repudiation of receipt (NRR) can be easily added to the basic protocol. Before sending K_O^{i-1} to R, S can ask R for an NRO token for "NRR"$|m$, which is then passed on to O. This is illustrated in Figure 2. The NRR token consists of:

$$(R, \text{"NRR"}|m, j, K_R^j)SK_S, K_R^{j-1}, r$$

Square parentheses ([ ]) indicate that the message contained within them is sent via a confidential channel. As this protocol is just two interleaved instances of the basic NRO protocol, it still guarantees that O and R can repudiate all forged NRO and NRR tokens, respectively. Note that this protocol actually implements *fair-exchange* of the NRO token for m and its NRR token, based on S as a trusted third party. If S behaves dishonestly, no fairness can be guaranteed: O might not receive the NRR token or R might not receive the NRO token.

The protocol as depicted in Figure 2 allows the possibility that R may refuse to send the NRR token after having received the candidate NRR token from S (from which R can extract m). An alternative approach is to include only a commitment to the message m in the candidate NRO token instead of the actual message itself. However, R has to trust that S will in fact send m *after* R has already acknowledged having received it. Note that if O and R happen to use different signature servers, additional inter-server message flows will be necessary.

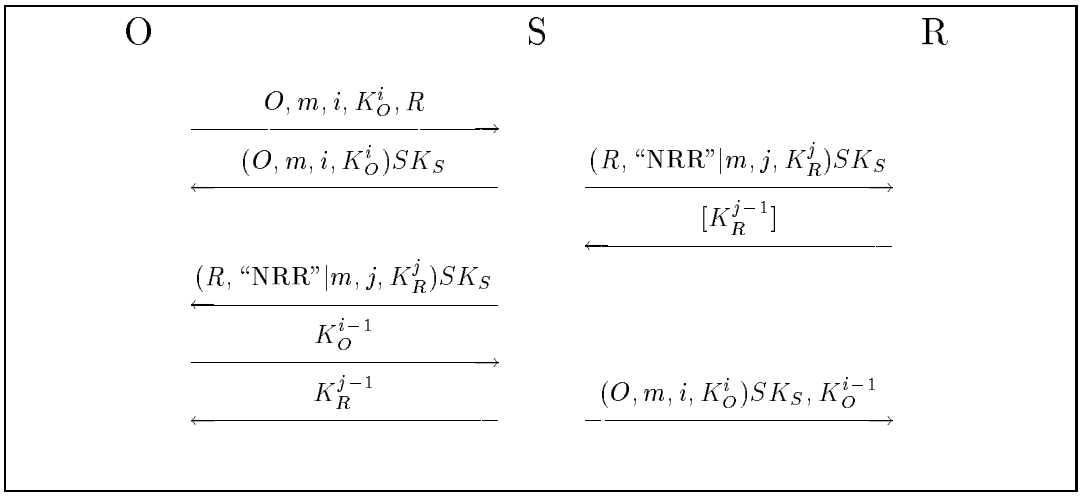

Figure 2: Protocol providing non-repudiation of origin and receipt.

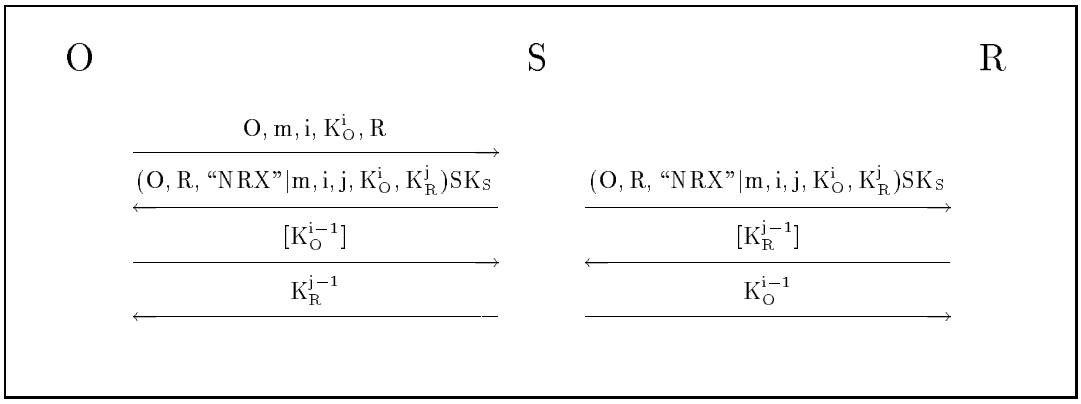

Figure 3: Protocol providing integrated non-repudiation of origin and receipt.

This protocol allows the possibility that either the NRO token or the NRR token may be optional, at the cost of an extra signature by S. The entire protocol has eight message flows. Further, the NRO and NRR tokens are linked only by the hash of the message. In environments where both NRO and NRR are mandatory, a modified protocol as shown in Figure 3 can be used. It results in a combined NRO and NRR token:

$$(O, \text{“NRX”}|m, i, j, K_O^i, K_R^j)SK_S, K_O^{i-1}, K_R^{j-1}$$

The modified protocol has only seven message flows and requires only a single signature by S.

5 Variations on the Theme

5.1 Reducing Storage Requirements for Users

In order to deny forged non-repudiation tokens, O has to store *all* signatures received from S, which might be a bit unrealistic for a device that is not even able to compute signatures. One can easily avoid this storage

problem by including an additional field H in S's signature that serves as a commitment on all the previous signatures made by S for that hash chain; i.e., an NRO token looks like

$$NRO^i := ((O, m_i, i, K_O^i, H^i)SK_S, K_O^{i-1})$$

The value H^i is recursively computed by $H^n := Cert_O$ and $H^{i-1} := f(H^i, NRO^i)$. The function $f()$ is a collision-resistant one-way hash function[2]. NRO^i is an NRO token on message m_i using the token public key K_O^i.

O has to store only the last value H^i and the last signature received from S. S has to store all signatures, and has to provide them to O in case of a dispute. If S cannot provide a sequence of signatures that fits the hash value contained in the last signature received by O, the arbiter allows O to repudiate all signatures and assumes that S cheated.

This idea of chaining previous signatures was used by Haber and Stornetta [3] for the construction of a time stamping service, based on the observation that the sequence of messages in H^i cannot be changed afterwards. One can combine their protocols with ours, using S as a time stamping server, as explained in Section 6.1.

5.2 Increasing Robustness

As mentioned above, a signature server must sign exactly one message for a given user per public key (K_O^i) in the hash chain. However, anyone can send a signature request in the form of the first flow, i.e., (O, m, i, K_O^i).

If the signature server does not subsequently receive the corresponding pre-image of the current public key (K_O^{i-1}), the current public key is rendered invalid in any case. This implies that an attacker can succeed in invalidating an entire chain of a user by generating fake signature requests in her name.

An obvious solution would be to require O and S to share a secret key to be used for computing (and verifying) a message authentication code over the first protocol flow.

An alternative solution is to give users the ability to invalidate token public keys without having to create a new chain. The construction is only slightly more complicated than the basic protocol: instead of one chain, each user generates *two* chains (computed with two **different** hash functions): $K_O^n, \ldots, K_O^0$ and $\hat{K}_O^n, \ldots, \hat{K}_O^0$.

Each token public key is now a pair of hash values, say, $(K_O^i, \hat{K}_O^j)$. If O receives the candidate token $(O, m, i, K_O^i, \hat{K}_O^j)SK_S$, she can either *accept* or *reject* it;

- O accepts by revealing K_O^{i-1}. The next token public key is $(K_O^{i-1}, \hat{K}_O^j)$.
- O rejects by revealing $\hat{K}_O^{j-1}$. The next token public key is $(K_O^i, \hat{K}_O^{j-1})$.

On receiving K_O^{i-1} or $\hat{K}_O^{j-1}$, server S creates

- the **non-repudiation token** $(O, m, i, j, \hat{K}_O^j, K_O^{i-1})SK_S$ or
- the **invalidation token** $(O, \text{"INV"}|m, i, j, K_O^i, \hat{K}_O^{j-1})SK_S$

respectively.

The additional signature by S is necessary because for one signature

$$(O, m, i, j, K_O^i, \hat{K}_O^j)SK_S$$

it can easily happen that both K_O^{i-1} and $\hat{K}_O^{j-1}$ become public, i.e., the combination of the first signature with one pre-image would not be unambiguous and recipient R could not depend on what he receives. Instead of making two signatures, S can instead include two commitments $h(a_{NRO})$ and $h(a_{INV})$ of two random numbers a_{NRO} and a_{INV} in the first signatures. Then, S can release one of the two random numbers to O. The random number together with the first signature serves as either the NRO token or the invalidation token. Note that a cheating S could generate both tokens for the same token public key, but O could easily prove that S cheated by showing the token received.

[2]Note that $f()$ may be the same as $h_O()$. However, collision-resistance is a mandatory property for $f()$ (if S succeeds in breaking the collision-resistance of $f()$, it can forge signatures which O may not be able to refute since O no longer retains all past signatures). As we mentioned in Section 3.5, collisions in $h_O()$ are not equally catastrophic from the point-of-view of O.

5.3 Support for Roaming Users

In the basic protocol, the trust placed on the signature server is quite limited — it is trusted only to protect its secret key from intruders and to generate signatures in a secure manner. This limited trust enables a mobile user to make use of a signature server in foreign domains while travelling. Normally the signature server in the user's home domain will be in charge of the user's hash chain. Whenever the user requests to be transferred to a signature server in a different domain, an agreement could be signed by the user and the old signature server authorising the transfer of charge of the user's hash chain. As usual, the pre-image of the current token public key, used to sign this agreement will become the next token public key.

In other words, instead of having a single root public key certificate (which includes the identity of the "home" signature server), a chain of public key certificates could be used. The chain consists of the root public key certificate signed by the home CA and one *hand-off certificate* every time the charge for the user's public key has changed hands:

$$(O, n, K_O^n, S_0)SK_{CA}$$

$$(O, n_l, K_O^{n_l}, S_l)SK_{S_{l-1}},\ for\ 0 < n_l < n, l > 0$$

where, $S_0 = S$ and n_l is the index of the token public key used to sign the request for the l^{th} hand-off (from S_{l-1} to S_l).

To effect a change in charge during a handoff, the following procedure is carried out:

1. The user O sends a hand-off request to both the current signature server S_{l-1} and the intended signature server S_l. As this request must be non-repudiable, this step is essentially a run of the basic protocol to generate a NRO token with a message that means "hand-off from S_{l-1} to S_l requested." S_{l-1} will issue a candidate NRO token for the request using the current token public key $K_O^{n_l}$ and O will validate the token by revealing $K_O^{n_l-1}$).

2. When the NRO token is received and verified by S_{l-1}, it generates a corresponding hand-off certificate described in the previous paragraph and sends it to both O and S_l. It will no longer generate signatures on behalf of O for that hash chain unless charge is explicitly transferred back to it at some point. In addition, it will store both the hand-off certificate and the corresponding NRO token.

3. When S_l has received both the NRO token and the hand-off certificate, it will be ready to generate signatures on behalf of O, starting with $K_O^{n_l-1}$ as the first token public key.

5.4 Key Revocation

As with any certificate-based system, there must be a way for any user O to revoke her hash chain.[3] If the currently secret portion of O's hash chain (say K_O^i, for $i = p-1, p-2, \ldots 1$) has been compromised, O will detect this when she attempts to construct an NRO the next time for the token public key K_O^p: S will return an error indicating the current token public key $K_O^q (q < p)$ from S's point of view. O can attempt to limit the damage by doing one of the following:

1. invalidate all remaining token public keys $K_O^i (i = q, q-1, \ldots 1)$ by requesting NRO tokens for them, or

2. notifying S to invalidate the remaining hash chain by sending it a non-repudiable request to that effect and receiving a non-repudiable statement from S stating that the hash chain has been invalidated. This can be implemented similar to the invalidation tokens described in Section 5.2 — except in this case the token would invalidate the entire chain and not just a single key.

[3] Revocation by authorities is not an issue in this system because the user has to interact with the signature server for the generation of every new NR token anyway.

5.5 General Signature Translation

In a more general light, the signature server in S^3 can be viewed as a "translator" of signatures: *it translates one-time signatures based on hash-functions into traditional digital signatures.* The same approach can be used to combine other techniques such that the result provides some features that are not available from the constituent techniques by themselves.

For example, one could select a traditional digital signature scheme (say D_1) where signing is easier than verification (e.g. DSS) and one (say D_2) where verification is easier than signing (e.g. RSA with a low public exponent) and construct a similar composite signature scheme. The signature key of an entity X in digital signature scheme D is denoted by SK_X^D. To sign a message m, an originator O would compute $(m)SK_O^{D_1}$ and pass it along with m to the signature server S. If the server can verify the signature, it will translate it to $(m,(m)SK_O^{D_1})SK_S^{D_2}$. In other words, the composite scheme allows digital signatures where both signing and verification are computationally inexpensive.

6 Applications

6.1 Building a Secure time stamping Service

In Section 3.5, we argued that S^3 meets the standard requirements of a signature scheme. The structure of the non-repudiation tokens result in an additional property: non-repudiation tokens issued by a given user have a strict temporal ordering among them.

Recall the structure of the non-repudiation tokens described in Section 5.1:

$$NRO^i := ((O, m_i, i, K_O^i, H^i)SK_S, K_O^{i-1})$$

where,

$$H^n := K_O^n, \quad H^{i-1} = f(H^i, NRO^i)$$

If $f()$ is collision-resistant, the chaining factor H^i imposes an order among the messages signed. We call this a *token chain*. Suppose that

$$\{NRO^i\}, i = n \ldots p, p-1 \ldots q$$

indicates the token chain of (NRO tokens issued by) a certain user O at a given time. It is easy to see that all $NRO^q, q < p$, must have been created after NRO^p. As NRO^p is a commitment on m_p, it follows that O knew m_p and showed it to S before NRO^q was created. O and S, either by themselves or in collusion, cannot create NRO^p after NRO^q was created. This enables us to build a time stamping service based on S^3.

Ideally, a time stamping system must be able to impose a total order on all the messages time stamped. We can adapt the approach used in [3], where S generates a chaining factor from a *single*, global chain. Every signature generated by the server has a chaining factor from this global chain. To verify a given time stamp, one needs to know the owners of the previous (and if necessary the subsequent) time stamps generated by the time stamping server.

In Section 4, we outlined a protocol that results in a combined NRO/NRR token. Chaining factors can be included in this token as well. The resulting NR token will be

$$NR_{AB}^{ij} = (A, \text{"NRX"}|m_{ij}), i, j, K_A^i, K_B^j, H_A^i, H_B^j)SK_S, K_A^{i-1}, K_B^{j-1}$$

where

$$H_A^n := K_A^n, \quad H_A^{i-1} = f(H_A^i, NRO_A^i)$$

$$H_B^m := K_B^m, \quad H_B^{j-1} = f(H_B^j, NRO_B^j),$$

and NR_{AB}^{ij} is the same as NRO_A^i and NRO_B^j. Each time A sends a message to B using the modified protocol resulting in a combined NRX token, the token chains of A and B are "synchronised:" any $NRO_A^p, p > i$ must have been created before any $NRO_B^q, q < j$. Although this does not necessarily result in a total order, the more chains are synchronised after a message has been signed, the greater the number of witnesses to the time of signing.

Thus, S^3 can be used to temporally link the tokens generated by S on behalf of multiple users. A practical implementation of a time stamping service can be constructed by requiring that S include a time stamp in each signature generated. The aim of this construction is not to provide an absolute guarantee that the time stamp in a document is precisely correct. Instead, the temporal ordering property of S^3 signatures is used to verify if the time stamp is plausible. In case of a dispute about the time stamp on a signature by A, the token chains of all parties synchronised to A's chain after the signature was made are examined (suppose there are n such parties). If these token chains satisfy all the temporal ordering relationships discussed above *and* if there is a sufficient number of honest parties among those linked to A's token chain, then the time stamp is probably correct. In this scenario, at least all but one of the $n+2$ parties involved must collude in order to produce a fake time stamp which cannot be proved to be a fake.

When digital signatures are used as a means to provide accountability, it is crucial to have unforgeable time stamps embedded in the signatures. The usual technique to achieve this is to use a seperate, external time stamping service is used in conjunction with a traditional digitial signature mechanism. The structure of S^3 makes it a signature scheme with an integrated unforgeable time stamping facility.

6.2 Applications with a Fixed Recipient

As the role of the signature server is verifiable, the recipient can also play that role. This is useful in applications where several non-repudiable messages need to be sent to the same fixed recipient. An example of this is a home-banking" (or electronic funds transfer) application, where customers send signed payment orders to their bank.

Payments messages are of the form

$$m = (payee,\ amount,\ date)$$

When a payer wants to make a payment, he constructs a message of the above form and executes the normal S^3 protocol with the bank, resulting in an NRO token for the message. The bank then transfers *amount* to the account of *payee* and issues a special NRR token, which can be its signature on the entire NRO token. Optionally, it may also get a S^3 NRR token from the payee and forward it to the payee. The non-repudiation tokens serve as evidence of the transaction.

The idea of using a hash chain for repeated, *fixed-value* payments was suggested recently [9, 4]. We have been able to use S^3 for payments of arbitrary values because S^3 provides non-repudiation of origin for arbitrary messages.

7 Analysis

Computation: Ordinary users of S^3 need to be able to compute one-way hashes and to verify traditional digital signatures. Only the signature servers and CAs are required to generate traditional digital signatures. Key generation for ordinary users is also relatively simple: the user needs to be able to generate a random number. In contrast, key generation in traditional digital signature systems is typically more complex, involving, for example, the generation of large prime numbers. In summary, the computational requirements for ordinary users of S^3 are less than those using a traditional digital signature scheme offering comparable security.

Storage: Using the improvement described in Section 5.1, users need to store only the last signature received from S, the pre-image of the current token public key and the sequence number, and the public keys needed to verify certificates.

Signature servers need to store all generated signatures in order to provide them to the users on demand. The stored signatures are necessary only in case of a dispute. Therefore, they can be periodically down-loaded to a secure archive.

Communication: The communication overhead of S^3 is comparable to that of standard symmetric non-repudiation techniques because a third party, S, is involved in each generation of a non-repudiation token.

Using traditional digital signatures, the involvement of third parties can be restricted to exception handling, whereas token generation is usually non-interactive. The price to be paid for this gain in efficiency is

that revocation of signature keys becomes more complicated. Note that in S^3, revoking a key is trivial. O simply has to invalidate the current chain.

Security: In the preceding sections, we demonstrated that as long as the registration procedure, the digital signature scheme, and the one-way hash function are secure, both users and signature servers are secure with respect to their respective objectives. Furthermore, the security of originators depends on the strength of the hash function and not on the security of the digital signature scheme.

Additionally, we observe that in practice, traditional digital signature algorithms are not applied directly to arbitrarily long messages. Instead, a collision-resistant, one-way hash function is first applied to the message to produce a fixed-length digest or fingerprint which is then signed using the signature algorithm. The overall security therefore depends on both the traditional digital signature algorithm and the hash function. Signature servers typically have significantly more computational resources available to them than do ordinary users. Hence they can choose a higher grade security (e.g. much longer signature keys) from a given digital signature algorithm. Thus, S^3 gives ordinary users the ability to produce stronger signatures than they could have been able to by using traditional signatures by themselves in the standard way.

8 Related Work

Although non-repudiation of origin and receipt are among the most important security services, only a few basic protocols exist. See [2] for a summary of the standard constructions. We are not aware of any previous work that aims to minimize the computational costs (at the protocol level) for ordinary users while providing the same security as standard non-repudiation techniques based on asymmetric cryptography.

The efficiency problem as addressed by specific designs of signature schemes was mainly motivated by the limited computing power of smart cards and smart tokens. [12] lists most known proposals. Typically they are based on pre-processing or on some asymmetry in the complexity of signature generation and verification (i.e., either sender or recipient must be able to perform complex operations, but not both). Note that although server-supported signatures use a signature scheme that is asymmetric with respect to signature generation and verification, ordinary users are *never* required to generate signatures; thus, both sender and recipient are assumed to be computationally weak.

In his well-known paper [6], Lamport proposed using hash chains for password authentication over insecure networks. There had been other, earlier proposals to use one-way hash functions to construct signatures. Merkle has presented an overview of these efforts [7]. The original proposals in this category were impractical: a proposal by Lamport and Diffie requires a "public key" (i.e. an object that must be bound to the signer beforehand) and two hash operations to sign *every* bit. Using an improvement attributed to Winternitz involving a single public key (which is the n^{th} hash image of the private key) and n hash operations, one can sign a single message of size $log_2 n$ bits. Merkle introduced the notion of a tree structure [7]; in one version of his proposals, with just a single public key, it is possible to sign an arbitrary number of messages. Nevertheless, it took either a large number of hash operations or a large amount of storage in order to sign more than a handful of messages corresponding to the same public key.

Motivated by completely different factors, Pfitzmann et al. [10][11, Section 6.3.3] proposed a fail-stop signature protocol which uses the same ideas as S^3. There, the signature server is also the recipient of the signature (which is a sub-case in the scope of S^3), and the goal is to achieve unconditional security for the signer against the server (in the sense of fail-stop signatures). The protocol has a similar structure as the one in Section 1.[4] Because of the specific security requirements, all parties have to perform complex cryptographic operations, and signatures are not easily transferable.

[4] It uses a so-called bundling function $h()$ instead of the conceptionally simpler hash function used in S^3. A value $h(x)$ is used as O's current public key. To give a NRO token for message m to S, the signer O sends m to S, which answers by $(m, h(x))SK_S$. Finally O sends x to S, which terminates the protocol. The NRO token for S is $(m, h(x))SK_S, x$. The consumed public key $h(x)$ can be renewed by including a new public key $h(x')$ in m.

9 Acknowledgments

We thank Jay Black and the anonymous referees for their valuable comments on previous drafts of this paper, Michael Steiner and Ceki Gülcü for many insights and thought-provoking discussions, Lilli-Marie Pavka for helping us polish the presentation, and Didier Samfat for suggesting the original problem and piquing our interest.

References

[1] N. Asokan, G. Tsudik, and M. Waidner. Server-supported signatures. In Elisa Bertino et al., editors, *Proceedings of the Fourth European Symposium on Research in Computer Security (ESORICS)*, number 1146 in Lecture Notes in Computer Science, pages 131–143. Springer-Verlag, Berlin Germany, September 1996.

[2] Warwick Ford. *Computer Communications Security – Principles, Standard Protocols and Techniques.* Prentice Hall, New Jersey, 1994.

[3] S. Haber and W. S. Stornetta. How to time-stamp a digital document. In *Advances in Cryptology – CRYPTO '90*, pages 437–455. Springer-Verlag, Berlin Germany, 1991.

[4] Ralf Hauser, Michael Steiner, and Michael Waidner. Micro-payments based on ikp. Research Report 2791 (# 89269), IBM Research, Feb 1996.

[5] ISO/IEC JTC1, Information Technology SC 27. Information technology - security techniuqes - non-repudiation. ISO/IEC JTC 1/SC 27, 1996. Contains 3 parts; Current version dated June 1996; Next version expected in early 1997.

[6] Leslie Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, November 1981.

[7] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *Advances in Cryptology – CRYPTO '87*, number 293 in Lecture Notes in Computer Science, pages 369–378, Santa Barbara, CA, USA, August 1987. Springer-Verlag, Berlin Germany.

[8] NIST National Institute of Standards and Technology (Computer Systems Laboratory). Secure hash standard. Federal Information Processing Standards Publication FIPS PUB 180-1, April 1995.

[9] Torben P. Pedersen. Electronic payments of samll amounts. In *Cambridge Workshop on Security Protocols*, volume 1189 of *Lecture Notes in Computer Science*, pages 59–68, April 1996.

[10] Andreas Pfitzmann, Birgit Pfitzmann, and Michael Waidner. Practical signatures where individuals are unconditionally secure. Unpublished manuscript, available from the authors (pfitzb@informatik.uni-hildesheim.de), 1991.

[11] Birgit Pfitzmann. *Digital Signature Schemes — General Framework and Fail-Stop Signatures.* Number 1100 in Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1996.

[12] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* John Wiley & Sons, Inc, 1996.

[13] Douglas R. Stinson. *Cryptography: theory and practice.* CRC Press Series on Discrete Mathematics and Its Applications, edited by Kenneth Rosen. CRC Press, Boca Raton, Florida, 1995.

How to construct optimal one-time signatures

Kemal Bicakci [a], Gene Tsudik [b], Brian Tung [c]

[a] *Informatics Institute, Middle East Technical University, Ankara 06531, Turkey*
[b] *Computer Science Department, University of California, Irvine, CA 92612, USA*
[c] *USC Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90292, USA*

Received 25 March 2002; received in revised form 23 April 2003; accepted 24 April 2003

Responsible Editor: S.F. Wu

Abstract

One-time signature (OTS) offer a viable alternative to public key-based digital signatures. OTS security is typically based only on the strength of the underlying one-way function and does not depend on the conjectured difficulty of some mathematical problem. Although many OTS methods have been proposed in the past, no solid foundation exists for judging their efficiency or optimality. This paper develops a methodology for evaluating OTS methods and presents optimal OTS techniques for a single OTS or a tree with many OTS's. These techniques can be used in a seesaw mode to obtain the desired tradeoff between various parameters such as the cost of signature generation and its subsequent verification.

Keywords: One-time signature; Digital signature; On-line/off-line digital signature; Hash function; Combinatorics

1. Introduction

Modern networks and Internetworks are more open than ever before, in an attempt to make information available on a ubiquitous basis. Networks are also faster than before, with available bandwidths measured in Gb/s. However, instead of alleviating congestion on the information highway, this has only encouraged the transmission of greater numbers of large data objects, especially with the recent popularity of multimedia presentations, voice- and video-conferencing, and large-scale scientific computing. The composition of network traffic has changed from yesterday's text files to today's enormous datasets produced by sophisticated remote visualization and rendering tools.

These developments make it important to maintain data integrity and privacy in a manner that is both highly secure and efficient. Traditional digital signature methods based on public key cryptography are simply untenable from a performance perspective. Furthermore, the security of public key cryptosystems (e.g., RSA or DSS [1,2]) is based on complex mathematical problems, such as factoring or discrete logarithms. The mathematical basis is both a blessing and a curse: the former because it lends itself to simple and elegant

design, and the latter because there is no assurance that there are no efficient algorithms for solving the underlying mathematical problems.

One-time signature (OTS) provide an attractive alternative to public key-based signatures. Unlike signatures based on public key cryptography, OTS is based on nothing more than a one-way functions (OWFs). [1] Consequently, OTSs are claimed to be more efficient since no complex arithmetic is typically involved in either OTS generation or verification. In practice, security of traditional public key-based digital signatures is based on two factors: conjectured-hard mathematical problems and the message digest function used to produce a fixed-size digest from arbitrarily long input data. (A secure message digest function suitable for this purpose must be both one-way and collision-resistant.) Using OTSs essentially allows us to eliminate the first factor altogether.

The OTS concept has been known for over two decades. It was initially developed by Lamport [6] and subsequently enhanced by Merkle [7] and Winternitz [8]. Bleichenbacher et al. [9–11] formalized the concept of OTS using directed acyclic graphs (DAGs).

In the simplest case, a message signer prepares an OTS by first generating a random number r which serves as a one-time private key. The signer then securely distributes a one-time public key $h(r)$, where $h(\cdot)$ is a suitable collision-resistant OWF. This public key, sometimes also referred to as an *anchor value*, is later used by the signature verifier(s) to verify the signature.

A signature is constructed by revealing the one-time private key r. A receiver (verifier) that obtains r' (which may or may not be the same as r) checks that it could only be have been generated by the claimed signer by computing $h(r')$. If this value matches the one-time public key $h(r)$, the OTS is considered valid. This, in effect, allows the signing of a *predictable* 1-bit value and provides one-time origin authentication. In order to sign *any* 1-bit value, two random numbers $\{r_0, r_1\}$ are needed. This way, both $h(r_0)$ and $h(r_1)$ are pre-distributed but at most one of $\{r_0, r_1\}$ is revealed as part of a signature. The pair $(r_0, h(r_0))$ represents an OTS of message "0", whereas $(r_1, h(r_1))$ is an OTS of "1".

Merkle extended this method to allow the signing of an arbitrary message. It begins by reducing the message to a fixed-length quantity using a collision-resistant message digest function, as is customary with traditional public key signatures. However, instead of transforming this quantity with a private key, each bit has an associated OTS and the signature for the entire message is represented as the concatenation of the OTS for each "1" bit in the message digest, along with some extra values to ensure that this per-bit signature is not itself modified.

As stated, this algorithm requires the one-time public keys for the OTSs to be distributed in a secure fashion. Since this is typically done using public key methods, the benefit of using efficient OTSs is apparently lost. However in [7], Merkle also introduced a scheme where these signatures are embedded in a tree structure, allowing the cost of a single public key signature (to sign the initial anchor values) to be amortized over many OTSs. In this formulation, signatures are longer, by at most an order of magnitude. However, the extra length (which was a concern two decades ago) is negligible today owing to the high speed of modern networks.

Despite their performance advantage and increased security, OTSs have remained on the periphery of security research since their inception. In particular, no practical evaluation of OTS capabilities has been done. This open issue is precisely the topic of the present paper. In order to obtain better understanding of OTS optimality, we first address a more general issue of how to maximize the message size (of a message to be signed) while minimizing the number of random quantities to be used in OTS generation (and, hence, the number of OWF operations). Our result leads us towards an optimal OTS construction where efficiency corresponds to the smallest number of OWF operations used in both generation and verification of an OTS. We then amend this definition of efficiency to take into account situations where multiple verifications are necessary, e.g.,

[1] Examples of conjectured OWFs include DES [3], MD5 [4], and SHA [5]. There is strong (albeit, folkloric) evidence as to the existence of true OWFs.

with multi-destination e-mail or, more generally, secure multicast. This leads us to consider a slightly different notion of optimality.

The outline of the paper is as follows. After introducing Merkle's signature algorithm in Section 2, we generalize the OTS constructions and present our optimal technique in Section 3. We evaluate the performance of our OTS construction in directed acyclic computation graph notation in Section 4. Section 5 is for describing how to encode a message for the signature using our technique. We make the cost analysis of a single OTS or a tree with many OTSs in Sections 6 and 7, respectively. In Section 8, we present a method to construct the optimal tree, more precisely we show how to choose the optimal depth of this tree. Section 9 discusses practical implementation aspects and possibilities for future work and Section 10 concludes this paper.

2. Merkles one-time signature construction

One notable and efficient OTS construction is due to Merkle [12]. (Others can be found in [13,14].) Assuming input messages of size b, let $s = (\lfloor \log b \rfloor + 1)$ and let $n = b + s$. The signer generates a secret key vector of size $(b + 2s)$ of random numbers:

$$R = \{R_1, \ldots, R_b, L_{1,0}, L_{1,1}, \ldots, L_{s,0}, L_{s,1}\}.$$

The signer then applies the OWF to each element of the secret key vector and distributes the resulting public key vector to the intended verifier(s):

$$H(R) = \langle H(R_1), \ldots, H(R_b), H(L_{1,0}), H(L_{1,1}), \ldots, H(L_{s,0}), H(L_{s,1}) \rangle.$$

Subsequently, to sign a b-bit message m, the signer counts the number of "1" bits in m, encodes the count as an s-bit string and appends it to m. The result is an n-bit message m'. The actual signature $\mathrm{SIG}(m)$ is constructed as follows:

for $i = 1$ to b **do begin**
  if $(m[i] == 1)$ **then** /* ith bit of m' is "1" */
    release $H(R_i)$
end /* for */
for $i = (b + 1)$ to n **do begin**
  if $(m[i] == 1)$
    release $H(L_{i,1})$
  else
    release $H(L_{i,0})$
end /* for */

For example, if $b = 4$ (thus, $n = 7$) and $m = 0101$, then $m' = 0101010$ and $\mathrm{SIG}(m) = \{R_2, R_4, L_{1,0}, L_{2,1}, L_{3,0}\}$. The verifier checks the signature by applying H to each element of $\mathrm{SIG}(m)$ and checking it against the public key vector $H(R)$.

To summarize the cost of Merkle's OTS construction, the signer generates $(b + 2s)$ random numbers and performs as many OWF computations. Each verifier performs, on the average, $(b/2 + s)$ OWF computations. For example, for a 160-bit message (e.g., an SHA1 digest), 176 and 88 OWF operations are needed to sign and verify, respectively.

Despite its relatively low cost and simplicity, the above is basically an ad hoc construction. No argument for its optimality has been provided in Merkle's work. Moreover, it remains unclear what optimality means in the context of an OTS system.

3. One-time signature generalization

More generally, a message sender prepares a signature by generating an n-element random number vector $R = (r_1, r_2, \ldots, r_n)$. He then computes $H(R) = \langle H(r_1), H(r_2), \ldots, H(r_n) \rangle$ where $H(\cdot)$ is a suitable OWF. The sender then securely distributes the one-time public key vector $H(R)$ to all intended verifiers.

Signature generation is the process of mapping the input message into a subset $S \subset R$. S is then attached to the message as its signature. To verify S, each receiver computes a similar mapping from the input message into a subset T of $H(R)$. The signature is considered valid only if $T = H(S)$.

The mapping function must satisfy a condition which we refer to as *incomparability:* for any message D_1, an attacker must be unable to find another message D_2 such that $F(D_2) \subset F(D_1)$ where $F(D_i)$ corresponds to signature subset S_i for the message D_i. Otherwise, if the legitimate signer distributes $\langle D_1, F(D_1) \rangle$, the attacker could replace

D_1 with D_2, reduce the set $F(D_1)$ to $F(D_2)$, and obtain a valid message–signature pair $\langle D_2, F(D_2)\rangle$.

This leads us to ask the question: *When R contains n random numbers, how many distinct messages can be signed or in other words what would be the maximum size of the message space?*

For $n = 1$, the answer is one, and the signature is one random number. For $n = 2$, the answer is two: the signature can be either r_1 or r_2. If we were to map a message onto the signature subset $\{r_1, r_2\}$, that choice would eliminate any other subset, allowing a single distinct message.

In general, we observe that, for any n, we can obtain a valid message mapping by drawing from all subsets containing $p < n$ random numbers. Clearly, no one such subset can be the subset of another, allowing us $C(n,p) = n!/p!(n-p)!$ distinct messages.

In [15], it is shown that for any n, the domain of mapping M is greatest when $p = \lfloor n/2 \rfloor$. This allows us to sign any one of

$$B_n = \binom{n}{\lfloor n/2 \rfloor} \tag{1}$$

distinct messages, i.e., we are able to sign an arbitrary $(\log B_n)$-bit message. For example, if R contains four elements 1, 2, 3, and 4, then the largest valid message set of R is

$$V = \{\{1,2\},\{1,3\},\{1,4\},\{2,3\},\{2,4\},\{3,4\}\}$$

which contains $B_4 = 6$ elements.

By inverting this formula, and using Stirling's approximation, we can see that to represent 2^b distinct values, n must satisfy

$$B_n > 2^b,$$

$$\frac{\sqrt{2\pi n}(n/e)^n}{[\sqrt{\pi n}(n/2e)^{n/2}]^2} > 2^b,$$

$$2^n\sqrt{2/\pi n} > 2^b$$

or, after taking base 2 logarithm of both sides,

$$n - \lg\sqrt{\pi n/2} > b. \tag{2}$$

For $b = 128$ (e.g., MD5), n must be at least 132, and each subset can be as small as size 64, since $C(132,64) > 2^{128}$. For $b = 160$ (e.g., SHA1), n must be at least 165 ($n = 164$ is just barely insufficient), with subsets of size 75.

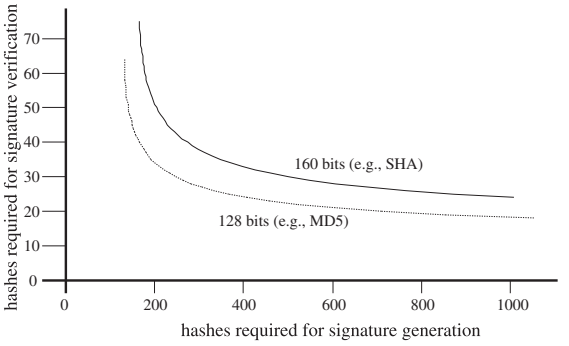

Fig. 1. Signature generation/verification hash profile.

Note that we can freely increase n and decrease p, or similarly, decrease n and increase p, provided $C(n,p) > 2^b$ and the signer and verifier(s) decide beforehand on the values of n and p. At one extreme, as we have shown, there is the lowest n such that there exists a p so that $C(n,p) > 2^b$. At the other extreme, one could choose $n = 2^b$, and allow $p = 1$; this would correspond to the case where there is a random number associated with each possible message, and the sender simply picks the appropriate one for each message to be signed! (This is clearly an intractable storage problem.)

In Fig. 1, we show the number of random numbers n versus the number of hashes required for verification p, for two popular message (digest) sizes.

We also observe that, for a valid (n,p) pair satisfying $C(n,p) > 2^b$, if the anchor values $H(R)$ are encrypted, the release of a subset in V containing p numbers can be used to exchange b bits of confidential information, since no one observing the p numbers can distinguish them from any others, or verify them, without being able to decrypt $H(R)$.

4. Efficiency assessment using directed acyclic graphs

Bleichenbacher et al. [9–11] formalized the concept of OTS using DAGs. They observed that the structure of the OTS computation leading

from the secret key components to the public key can be represented as a DAG $G = (V, E)$ with vertex set V and and edge set E, where the vertices correspond to the secret key, the intermediate results, and the public key, and where a directed edge (v_i, v_j) in E indicates that v_i is an input to the OWF and computation resulting in v_j.

In order to design a OTS based on a DAG, there are two important requirements. First, every OTS must be verifiable, i.e., the public key must be computable from it. Second, in order to prevent forgery, the set of signatures must satisfy the incomparability condition defined in previous section.

If we assume that all the public-key components are hashed in a binary tree to result in a single public-key component, then an efficient OTS algorithm can be formally defined as one in which the size of the message space is maximized while the size of the DAG is minimized. More precisely, if $\lg(\Gamma)$ is the number of message bits that can be signed, the efficiency of a one-time digital signature scheme Γ for a DAG G with $z = |V|$ vertices is given by

$$\eta(\Gamma) = \frac{\lg(\Gamma)}{z+1}.$$

In [10], the authors also presented their best graph construction, for which the efficiency is approximately equal to 0.473. We will now prove that our methodology provides a better construction in terms of their notion of efficiency.

In the previous section, we showed that the number of bits that can be signed using n random numbers is upper bounded by $n - \lg\sqrt{\pi n/2}$. In Fig. 2, we demonstrate that when we assemble the DAG of our construction with n random numbers, the number of vertices is equal to $2n + 1$. Then the efficiency is upper bounded by 0.5 as seen from Eq. (3).

$$\lim_{n\to\infty} \eta(\Gamma) = \frac{n - \lg\sqrt{\pi n/2}}{2n+2} = 0.5. \tag{3}$$

For $b = 128$ (e.g., MD5), this value is 0.4812 and for $b = 160$ (e.g., SHA1), it is 0.4819. We observe that both of them is better than the best construction in [10].

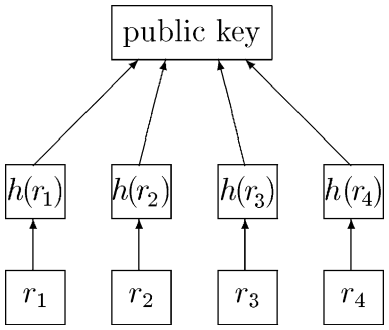

Fig. 2. The DAG representation of our construction for $n = 4$ ($z = 2n + 1 = 9$) where the public-key components are hashed in a binary tree to result in a single public-key component. To make the OTS verifiable, the signature contains the random numbers released as well as the hashes of unreleased random numbers.

Other than the efficiency concerns, we claim that our methodology is better than Bleichenbacher et. al.'s approach in two aspects:

- In practice, their proposed DAG construction is very complex and hard to implement whereas our combinatorial results are elementary and easy to grasp.
- In [10], the authors did not discuss how to encode a specific message for the DAG they used. In contrast, we provide an efficient method for encoding a message for signature in the next section.

5. Encoding a message for signature

Given a vector R of n random numbers, and a valid message set V of subsets each containing p of those numbers, we have shown that we can sign any one of $C(n, p)$ distinct messages. In this section, we describe the mapping M between messages and the elements of V, and demonstrate how to compute them efficiently.

Assume that the domain of M is composed of 2^b messages, and we have a way of representing each of the messages as a b-bit integer m. Let any subset S in V be expressed as $\{R_{a_1}, R_{a_2}, \ldots, R_{a_p}\}$. Arrange the subsets in V in lexically ascending order. For example, for $n = 4$, $p = 2$, the subsets are ordered

$\{R_1, R_2\}, \{R_1, R_3\}, \{R_1, R_4\}, \{R_2, R_3\},$
$\{R_2, R_4\}, \{R_3, R_4\}.$

Then the mapping $m = M(S, V)$ of each subset S is defined as the integer position of S in this list representation of V. For example, in the above case, $M(\{R_1, R_3\}, V) = 2$ and $M(\{R_3, R_4\}, V) = 6$. In general, for any n and p, the mapping of any subset $S = \{R_{a_1}, R_{a_2}, \ldots, R_{a_p}\}$, where $a_0 = 0$ and $a_1 < a_2 < \cdots < a_p$ is given by

$$M(S, V) = 1 + \sum_{i=1}^{p} \sum_{j=n-a_i+1}^{n-a_{i-1}-1} \binom{j}{p-i}. \tag{4}$$

Note that in order to compute the mapping for any subset, for a given n and p, we need only compute the binomial coefficients $C(j, p-i)$ for i from 1 to p, j from $p+1-i$ to $n-i$. Thus, each mapping requires $n - p - (n - a_p) = a_p - p$ additions.

Similarly, the mapping $S = M^{-1}(m, V)$ of a message represented by the integer m can be computed by subtracting binomial coefficients until zero is reached. This requires $a_p - p$ additions and comparisons. Pseudocode to do this conversion is as follows:

```
m_0 = m /* copy message to temporary value */
q = 1
for i = 1 to p do begin
   while m_0 > C(n - q, p - i) do begin
      m_0 := m_0 - C(n - q, p - i)
      q := q + 1
   end /* while */
   a_i := q
   q := q + 1
end /* for */
```

To put things in perspective, consider that a single MD5 hash computation requires approximately 500 arithmetic operations. Thus, our mapping (in both directions) costs less than one MD5 hash.

6. Cost analysis of a single one-time signature

6.1. All on-line case

In the preceding sections, we showed how to sign an arbitrary b-bit message using p of n random numbers. In this section, we will describe how to choose n and p to minimize the total cost of a OTS. Our initial assumption is that all of the signing process is performed on-line, once the message is presented.

The principal cost of generating a OTS (aside from the cost of securely distributing the anchor values $H(R)$) is the cost of computing $H(R)$; this costs n hashes. The principal cost of verifying a OTS (aside from the cost of verifying the anchor values) is the cost of computing $H(S)$; this costs p hashes. However, only a single sender generates a OTS, while potentially many receivers verify it. Thus, each hash involved in signature verification incurs a greater cost than one involved in signature generation.

In general, let each verification hash cost σ times as much as a generation hash. The total cost of a single signature is then proportional to $n + \sigma p$; this is the quantity that we shall try to minimize, subject to the condition that $C(n, p) > 2^b$.

We want to find n and p such that $C(n, p) \doteq C(n + \sigma, p - 1)$. As a first approximation, we have, from the definition of the binomial coefficient

$$\left(\frac{n}{n-p}\right)^{\sigma} \doteq \frac{n-p}{p}. \tag{5}$$

If we let $\alpha = p/n$, we have

$$(1-\alpha)^{\sigma} = \frac{\alpha}{1-\alpha}, \tag{6}$$

$$\alpha = (1-\alpha)^{\sigma+1}. \tag{7}$$

To find the optimal n and p, we find the p such that $C(\lfloor \alpha p \rfloor, p) > 2^b$.

6.2. On-line/off-line case

In [14], the authors introduce the new concept of on-line/off-line digital signature schemes. In these schemes the signing of a message is broken into two phases. The first phase is off-line. Though it requires a moderate amount of computation, it presents the advantage that it can be performed at leisure, before the message to be signed is even known. The second phase is on-line and it starts after the message becomes known. Since it utilizes

the precomputation of the first phase, it is much faster.

We observe that the signer can generate the vector $R = \{R_1, \ldots, R_n\}$, and by applying the OWF to each random number, he can generate the hashes off-line. The on-line phase is just a mapping and at the end of Section 5 we showed that it costs less than one MD5 hash operation. So, in this case we try to minimize the verification time. This is also due to the fact that many times only a single sender generates an OTS, but potentially many receivers verify it.

It is improper to take the verification time as only the time needed to make the mapping and generate the hashes of the random numbers. One of the disadvantages of OTS is its length; especially if we have a low bandwidth channel, the time needed to transmit the signature dominates the time for verification and cannot be neglected. Also, available bandwidth and computation power both vary over a wide range. We may therefore optimize n and p with respect to bandwidth and computation power.

Here, we will describe how to choose n and p to minimize the total time T needed to verify one OTS. Let's take the hash length as b and random number length as a. Assume a bandwidth of K bits/s and L seconds as the time required to perform one hash operation. Then

$$T = \frac{ap + bn}{K} + (p+1)L + m. \tag{8}$$

In the above formula, we ignore other delays such as queuing delays. The extra L is for generating the hash of the message, and m is the time to compute the mapping. The total time needed to verify one OTS is then proportional to $n + \sigma p$ where

$$\sigma = \frac{a + LK}{b}. \tag{9}$$

This optimization therefore reduces to the one analyzed in the previous section.

7. Amortized cost of many signatures

Using the OTS only once is inefficient, since the sender needs to sign the original hash image $H(R)$ using a conventional digital signature (e.g., DSS). Using Merkle's tree scheme [12] as illustrated in Fig. 3, we can sign an arbitrarily large number of messages with only one conventional signature. However, the incremental cost of generating and verifying an additional signature increases logarithmically with the number of signatures.

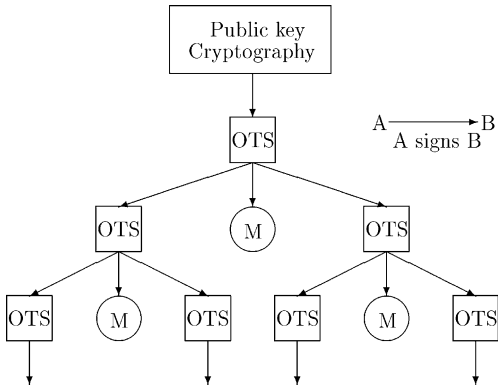

Fig. 3. Tree scheme for maintaining signature vectors.

In the tree scheme, one constructs a tree of signature nodes. Each signature node has a vector for signing each of its children as well as a single message. The root node is signed by conventional means—i.e., using a digital signature key.

One of the built-in features of Merkle's OTS tree construct is the ability to unambiguously order signatures. This is a very useful service in many application domains. For example, it is imperative in a military environment to establish strict causality of commands (and signed orders in general). Failure to do so can have disastrous consequences since re-ordered messages can lead to devastating mistakes on the battlefield. In the civilian realm, signature ordering is very important in electronic commerce, among other fields.

Consider a binary tree such that each node has three vectors and suppose that we choose n and p such that $C(n,p) > 2^b$. We would like to compute the cost, in hashes, of generating and verifying a signature at any given depth d.

To generate the signature, one needs to do a OTS of the message (requiring n hashes). Assuming the signer can cache the tree, no further computation is required.

Verification requires one to perform p hashes to verify the current node's signature of the message, and an additional p hashes to verify the parent node's signature of the current node. If the receiver has cached the tree, no further computation is required; otherwise, an additional $(d-1)p$ hashes are required.

In contrast to the single signature case, then, each of a sequence of signatures costs $n+2p$ hashes if receivers cache the signature tree, or $n+(d+1)p$ hashes if they do not. How reasonable is it to cache a signature tree? If we choose SHA as our message digest, we need to sign 160 bits of message, for which we could choose $n=165$ and $p=82$. Both signer and verifier must cache, for each node, $3n$ 160-bit numbers (the signer caches the random numbers R, the verifier caches the anchor values $H(R)$); this works out to 4950 bytes per node. This is easily supported.

In fact, receivers who cache the tree need only maintain the lowest layer of nodes, so that only about half the nodes already traversed need be kept at any time. They can prune additional information off the tree by removing the message signature vectors after they are exhausted.

8. Constructing the optimal tree

In Section 4, we showed how to choose n and p to minimize the total cost of an OTS. In this section, we will describe how to choose the parameters of a k-ary tree with a depth of d. Note in particular that the tree structure does not need to be binary. Also, we should stop somewhere in our tree; at some point the cost of using our large tree to verify an OTS is more than that of starting a new tree in which we have signed the root node by conventional means. In other words, we need to optimize the value of d.

We will try to minimize the average verification cost of one OTS. Suppose that the verifier only caches the root node of the tree. This is a reasonable assumption, especially when the signer uses the tree to sign messages for different receivers. Suppose also that the cost of verifying a traditional signature is C times more than verifying a OTS. In a k-ary tree with a depth of d, the number of messages that can be signed is

$$N=\sum_{y=1}^{d}k^{y-1}=\frac{k^d-1}{k-1}$$

and the number of OTS required

$$W=\sum_{y=1}^{d}yk^{y-1}=\frac{dk^{k+1}-(d+1)k^d+1}{(k-1)^2}.$$

In a single tree, the cost per message is

$$\frac{C+W}{N}=\frac{C+\sum_{y=1}^{d}yk^{y-1}}{\sum_{y=1}^{d}k^{y-1}}. \tag{10}$$

We try to find the smallest d such that the average cost per message is smaller than it is for $d+1$.

At optimal d,

$$\frac{C+\sum_{y=1}^{d+1}yk^{y-1}}{\sum_{y=1}^{d+1}k^{y-1}}\doteq\frac{C+\sum_{y=1}^{d}yk^{y-1}}{\sum_{y=1}^{d}k^{y-1}}$$

which is approximately equivalent to

$$C+(d+1)k^d+W\doteq k(C+W). \tag{11}$$

If we put W into its place and make necessary manipulations, our formula becomes

$$k^{d+1}\doteq(k-1)^2C+1.$$

Then, we can give the optimal tree depth d as

$$d\doteq\frac{\log[(k-1)^2C+1]}{\log k}-1. \tag{12}$$

In Fig. 4, we show the depth of a binary tree versus average normalized verification cost per message. One should choose the depth d where the average normalized cost per message is minimal.

Secondly, we would like to introduce a way to decide on the value of k. Suppose the signer caches the tree and S be the storage requirement for one random vector and its corresponding hash values. Then the storage cost per message is $(k+1)S$, which is independent of d. Suppose also that the storage cost per message is at most M. Then

$$k=\frac{M}{S}-1. \tag{13}$$

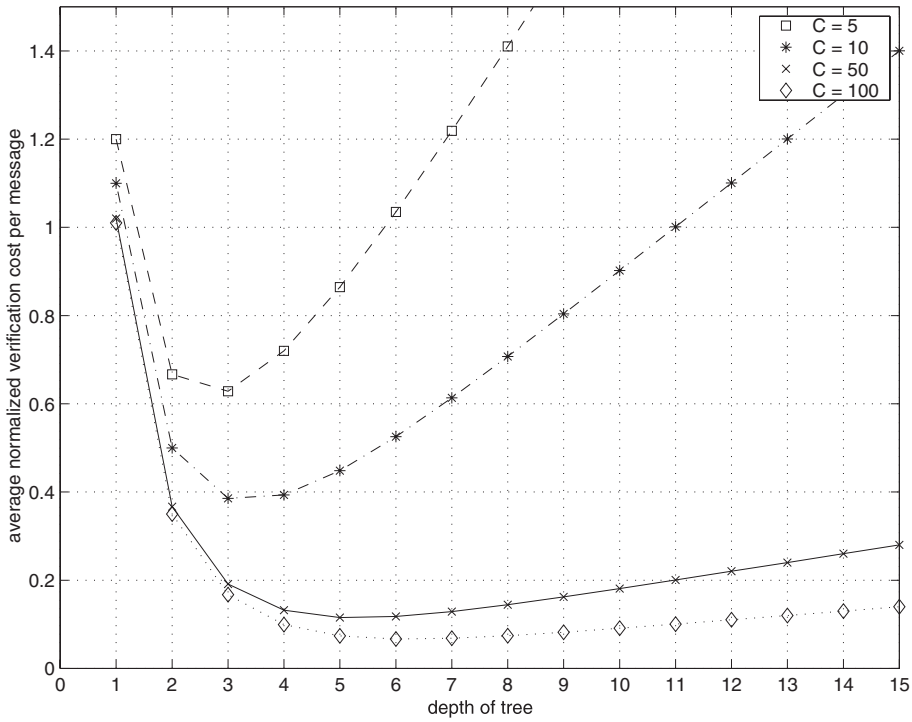

Fig. 4. Choosing the optimal depth for a binary tree.

So after estimating k with this simple formula, we can also find d by using the previous formula above.

9. Implementation and future work

In practical implementations, OTS use closely resembles that of public key signatures. For instance, in OTS, the sender distributes a certificate, which contains a public value, signed by a certification authority. The only difference is that the public value here is a sequence of hash function outputs (or a single output if these components are hashed in a single hash value), rather than a public key. Just as before, this certificate can be distributed in advance of the signed message, or it can accompany the signed message. Finally, the signature represents a transformation of a message digest of the overall message; the only differences are in the details of that transformation.

There is a clear analogy to traditional digital signature algorithms, which is summarized by the diagram below:

message digest $\Longleftrightarrow$ message digest

encryption with private key

$\Longleftrightarrow$ mapping to OTS random values

verification with public key

$\Longleftrightarrow$ verification via OTS anchors

Currently, we have implemented an OTS library which utilizes the suggested mapping and the tree structure. We will investigate the effect of changing the parameters to make it more efficient.

Subsequently, we will integrate our OTS library into specific applications that require fast digital signatures. One of the anticipated application domains is as an integrity mechanism in Active Networks. One of the basic tenets of the Active Networks concept is the use of so-called "smart packets," packets that carry the means for their own handling in routers and other network entities.

This paradigm immediately raises a number of security issues, data integrity and origin authentication being chief among them. For this reason, we maintain that ultra-fast digital signatures are an absolute must for Active Networks to be practical.

The results of our research will also be of interest to an intrusion detection system. Authenticating the source and contents of a response request is fundamental to the survivability of the systems at hand. At the same time, responses must also be executed in a timely fashion and not be allowed to queue indefinitely. We expect that OTS will provide a way for components of intrusion detection systems to quickly and efficiently establish the integrity of the messages they exchange.

10. Conclusion

We have provided a theoretical foundation for evaluating the efficiency and compactness of one-time digital signatures. This work reveals the absolute minimum complexity of computing a digital signature over a space of b-bit messages, based on the weighted costs of signature generation and verification. We have shown how this theoretical minimum can be achieved by using a simple and efficient mapping between messages and subsets of random numbers. We have demonstrated how this family of OTS schemes fits into an elegant protocol for amortizing the cost of OTSs over many messages. Finally, we have provided a theoretical foundation for evaluating the efficiency of the OTS tree structure based on the weighted costs of traditional and OTSs.

References

[1] R.L. Rivest, A. Shamir, L.M. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM 21 (2) (1978) 120–126.

[2] National Institute for Standards and Technology, Digital Signature Standard (DSS), Federal Register 56 (169), August 30, 1991.

[3] National Institute of Standards and Technology (NIST), FIPS Publication 46-2: Data Encryption Standard, December 30, 1993.

[4] R.L. Rivest, The MD5 message-digest algorithm, Internet Request for Comments, April 1992, RFC 1321.

[5] National Institute of Standards and Technology (NIST), FIPS Publication 180: Secure Hash Standard (SHS), May 11, 1993.

[6] L. Lamport, Constructing digital signatures from a one-way function, Technical Report CSL-98, SRI International, October 1979.

[7] R.C. Merkle, Secrecy, authentication, and public key systems, Technical report, Stanford University, June 1979.

[8] R.C. Merkle, A certified digital signature, in: G. Brassard (Ed.), Proceedings of the CRYPTO 89, Lecture Notes in Computer Science, vol. 435, Springer, Berlin, 1990, pp. 218–238.

[9] D. Bleichenbacher, U.M. Maurer, Directed acyclic graphs, one-way functions and digital signatures, in: Y.G. Desmedt (Ed.), Proceedings of the CRYPTO 94, Lecture Notes in Computer Science, vol. 839, Springer, Berlin, 1994, pp. 75–82.

[10] D. Bleichenbacher, U.M. Maurer, Optimal tree-based one-time digital signature schemes, in: Proceedings of the STACS 96, Lecture Notes in Computer Science, vol. 1046, Springer, Berlin, 1996, pp. 363–374.

[11] D. Bleichenbacher, U.M. Maurer, On the efficiency of one-time digital signatures, in: Proceedings of the ASIACRYPT 96, Lecture Notes in Computer Science, vol. 1163, Springer, Berlin, 1996, pp. 145–158.

[12] R.C. Merkle, A digital signature based on a conventional encryption function, in: C. Pomerance (Ed.), Proceedings of the CRYPTO 87, Lecture Notes in Computer Science, vol. 293, Springer, Berlin, 1988, pp. 369–378.

[13] J.N.E. Bos, D. Chaum, Provable unforgeable signatures, in: E.F. Brickell (Ed.), Proceedings of the CRYPTO 92, Lecture Notes in Computer Science, vol. 740, Springer, Berlin, 1992, pp. 1–14.

[14] S. Even, O. Goldreich, S. Micali, On-line/off-line digital signatures, in: G. Brassard (Ed.), Proceedings of the CRYPTO 89, Lecture Notes in Computer Science, vol. 435, Springer, Berlin, 1990, pp. 263–277.

[15] E. Sperner, Ein satz uber untermenge einer endlichen menge, Math. Z. 27 (1928) 544–548.

Kemal Bicakci received the B.S. degree in electronics engineering from Hacettepe University, Turkey in 1996 and the M.S. degree in electrical engineering (computer networks) from University of Southern California, USA in 1999. Between January 1998 and March 2000, he worked on various security projects as a graduate research assistant in USC Information Sciences Institute.

He is now a Ph.D. student in Informatics Institute, Middle East Technical University, Turkey. His research interests include network security and applied cryptography.

Gene Tsudik is an Associate Professor in the School of Information and Computer Science at UC Irvine. He has been active in the area of Internetworking, network security and applied cryptography since 1987. He obtained a Ph.D. in Computer Science from USC in 1991 for his research on access control in Internetworks. He then moved on to IBM Research (1991–1996) and USC Information Science Institute (1996–2000). Since 2000, he has been at UC Irvine.

Over the years, his research included: Internetwork routing, firewalls, authentication, mobile network security, secure e-commerce, anonymity, secure group communication, digital signatures, key management, ad hoc network routing, and, more recently, database privacy and secure storage. He has over 70 refereed publications and 7 patents.

Brian Tung is project leader and computer scientist at the USC Information Sciences Institute. He has been working in the security area for 8 years. During that time, he has led projects on intrusion detection and response, fast digital signatures using one-way hashes, and extensible watermarking, and he has also participated in a project to add public key cryptography to the Kerberos authentication system.

He is co-chair of the Common Intrusion Detection Framework (CIDF). In that position, he has co-edited and made significant technical contributions to the specification for the Common Intrusion Specification Language (CISL).

He received his B.S. in electrical engineering and computer science from UC Berkeley in 1989, and his M.S. and Ph.D. in computer science from UCLA in 1993 and 1994, respectively. His dissertation described the mathematics behind the use of simple finite state automata for optimization and autonomous control.

On Constructing Optimal One-Time Signatures

Kemal Bicakci

METU Informatics Institute, Inonu Bulvarı, 06531, Ankara, Turkey
bicakci@ii.metu.edu.tr

Brian Tung, Gene Tsudik

USC Information Sciences Institute
4676 Admiralty Way, Marina del Rey, 90292, CA, USA
brian@isi.edu, gts@isi.edu

Abstract. One-time signatures (OTS) offer a viable alternative to public key-based digital signatures. OTS security is based only on the strength of the underlying one-way function and does not depend on the conjectured difficulty of a mathematical problem. OTS methods have been proposed in the past but no solid foundation exists for judging their efficiency or optimality. This paper develops a new methodology for evaluating OTS methods and presents optimal OTS techniques for a single OTS or a tree with many OTSs. These techniques can be used in a see-saw mode to obtain desired tradeoff between various parameters such as the cost of signature generation and its subsequent verification and the cost of traditional signature verification and OTS verification.

1 Introduction

Digital signatures are rapidly becoming ubiquitous in many spheres of computing. Most current techniques for generating digital signatures are based on public key cryptography, e.g., RSA or DSS [12, 3]. These, in turn, are based on complex mathematical problems such as factoring or discrete logarithms. This solid mathematical basis is both a blessing and a curse: the former because it lends itself to simple and elegant design and the latter because there is no assurance that efficient algorithms do not exist for solving the underlying mathematical problems.

One-time signatures (OTS) provide an attractive alternative to public key-based signatures since---unlike signatures based on public key cryptography---OTS is based on nothing more than a one-way function (OWF). Examples of conjectured OWFs include DES [10], MD5 [11], and SHA [9]. There is strong (albeit folkloric) evidence as to the existence of true OWFs. Furthermore, OTSs are claimed to be more efficient since no complex arithmetic is typically involved in either OTS generation or verification.

The OTS concept has been known for almost two decades. It was initially developed by Lamport [4] and enhanced by Merkle [8] and Winternitz [7]. For the sake of brevity, we do not review the history of OTS research here; we defer instead to [6, 5] for a comprehensive treatment of the subject.

One efficient OTS construction is due to Merkle [6]. (Other efficient constructions can be found in [1] [2].) To summarize the cost of Merkle's OTS construction, the signer generates n = (b+log b) random numbers and performs n OWF computations. Each verifier performs, on the average, n/2 OWF computations which yields the average total of 1.5 * (b+log b). If we were to sign a 128-bit message (e.g., an MD5 digest) an average of 202 OWF operations would be necessary.

Despite its relatively low cost and overall elegance, this is basically an ad hoc construction. No argument for its optimality has been provided in Merkle's work. Moreover, it remains unclear what optimality means in the context of an OTS system. This open issue is precisely the topic of this paper. In order to obtain better understanding of OTS optimality, we first address a more general issue of how to maximize the message size (of a message to be signed) while minimizing the number of random quantities to be used in OTS generation (and, hence, the number of OWF operations). Our result leads us towards an optimal OTS construction where efficiency corresponds to the smallest number of OWF operations used in both generation and verification of an OTS. We then amend this definition of efficiency to take into account situations where multiple verifications are necessary, e.g., with multi-destination e-mail or, more generally, secure multicast. This causes us to consider a slightly different notion of optimality.

2 One-Time Signature Generalization

The question we are trying to find an answer is how many distinct messages can be signed when R contains n random numbers? For n = 1, the answer is one, and the signature is the one random number. For n = 2, the answer is two: the signature can be either r_1 or r_2. If we were to map a message onto the signature subset $\{r_1, r_2\}$, that choice would eliminate any other subset, allowing us only one distinct message. In general, we observe that, for any n, we can obtain a valid message mapping by drawing from all subsets containing p < n random numbers. Clearly, no one such subset can be the subset of another, allowing us C(n,p) = n!/p!(n-p)! distinct messages. In [13], it was shown that for any n, the domain of mapping M is greatest when we draw from all subsets containing $\lfloor n/2 \rfloor$ random numbers. This allows us to sign any one of:

$$B_n = \binom{n}{\lfloor n/2 \rfloor} \tag{1}$$

distinct messages, i.e., we are able to sign an arbitrary (log B_n)-bit message.
For example, if R contains four elements 1, 2, 3, and 4, then the largest valid message set of R is:

$$V = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}\}$$

which contains $B_4 = 6$ elements. By inverting this formula, using Stirling's approximation and taking the base 2 log of both sides, we can see that to represent 2^b distinct values, n must satisfy

$$n - \log\sqrt{\prod n/2} > b \tag{2}$$

For b = 128 (e.g., MD5), n must be at least 132, and each subset can be as small as size 64, since $C(132,64) > 2^{128}$. For b=160 (e.g., SHA1), n must be at least 165 (n = 164 is just barely insufficient), with subsets of size 75. Note that we can freely increase n and decrease p, or similarly, decrease n and increase p, as long as $C(n,p) > 2^b$. In Figure 1, we show the number of random numbers n versus the number of hashes required for verification p, for two popular message (digest) sizes.

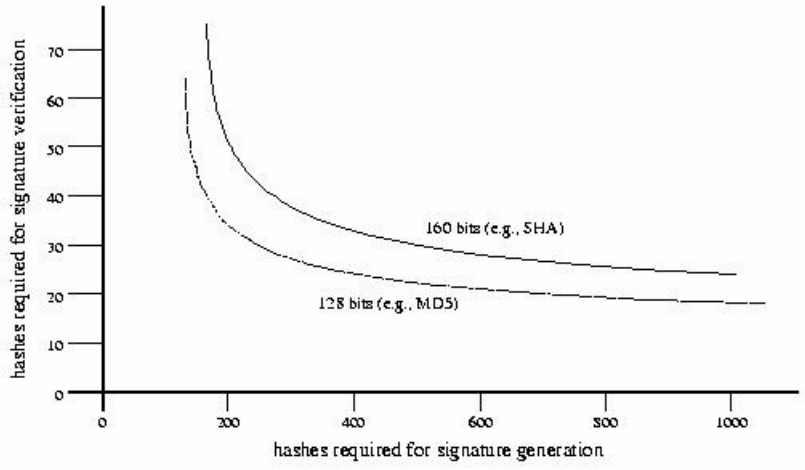

Figure 1: Signature generation/verification hash profile.

3 Cost Analysis of a Single One-Time Signature

3.1 All On-line Case

In the preceding sections, we showed how to sign an arbitrary b-bit message using p of n random numbers. In this section, we will describe how to choose n and p to minimize the total cost of a one-time signature. Our initial assumption is that all of the signing process is performed on-line, once the message is presented.

The principal cost of generating a one-time signature (aside from the cost of securely distributing the anchor values H(R) is the cost of computing H(R); this costs n hashes. The principal cost of verifying a one-time signature (aside from the cost of verifying the anchor values) is the cost of computing H(S); this costs p hashes. However, only

a single sender generates a one-time signature, while potentially many receivers verify it. Thus, each hash involved in signature verification incurs a greater cost than one involved in signature generation.

In general, let each verification hash cost σ times as much as a generation hash. The total cost of a single signature is then proportional to $n+\sigma p$; this is the quantity that we shall try to minimize, subject to the condition that $C(n,p) > 2^b$. We want to find n and p such that $C(n,p) \approx C(n+\sigma,p-1)$. As a first approximation, we have, from the definition of the binomial coefficient

$$\left(\frac{n}{n-p}\right)^{\sigma} \approx \frac{n-p}{p} \tag{3}$$

If we let $\alpha = p/n$, we have

$$\alpha = (1-\alpha)^{\sigma+1} \tag{4}$$

To find the optimal n and p, we find the p such that $C(\lfloor \alpha p \rfloor, p) > 2^b$.

3.2 On-line/Off-line Case

In [2], the authors introduce the new concept of on-line/off-line digital signature schemes. In these schemes the signing of a message is broken into two phases. The first phase is off-line. Though it requires a moderate amount of computation, it presents the advantage that it can be performed at leisure, before the message to be signed is even known. The second phase is on-line and it starts after the message becomes known. Since it utilizes the precomputation of the first phase, it is much faster.

We observe that the signer can generate the vector $R=\{R_1,...,R_n\}$ of n random numbers and by applying the OWF to each random number, he can generate the hashes off-line. The on-line phase is just a mapping and at costs less than one MD5 hash operation. So, in this case we try to minimize the verification time. This is also due to the fact that many times only a single sender generates an OTS, but potentially many receivers verify it.

It is improper to take the verification time as only the time needed to make the mapping and generate the hashes of the random numbers. One of the disadvantages of OTS is its length; especially if we have a low bandwidth channel, the time needed to transmit the signature dominates the time for verification and cannot be neglected. Also both available bandwidth and computation power vary in a wide range. We therefore need to decide on the values of n and p with respect to bandwidth and computation power.

Now, we will describe how to choose n and p to minimize the total time T needed to verify one OTS. Let's take the hash length as b and random number length as a.

Assume a bandwidth of K bits/sec. And L seconds as the time required to perform one hash operation. Then

$$T = \frac{ap + bn}{K} + (p+1)L + m \tag{5}$$

In the above formula, we ignore other delays such as queuing delays. One extra L is for generating the hash of the message and m is the time for the mapping. The total time needed to verify one OTS is then proportional to $n+\sigma p$ where $\sigma = (a + LK) / b$. Fortunately, this is the same quantity we have tried to minimize in the preceding subsection. So in this case we can also use the results we have obtained previously.

4 Amortized Cost of Many Signatures

Using the one-time signature only once is inefficient, since the sender needs to sign the original hash image H(R) using a conventional digital signature (e.g., DSS). Using a tree scheme, as in Merkle [6], we can sign an arbitrarily large number of messages with only one conventional signature. However, the incremental cost of generating and verifying an additional signature increases logarithmically with the number of signatures.

In the tree scheme, one constructs a tree of signature nodes. Each signature node has a vector for signing each of its children as well as a single message. The root node is signed by conventional means---i.e., using a digital signature key. In this treatment, we consider only a binary tree, so that each node has three vectors. Suppose that we choose n and p such that $C(n,p)>2^b$. We would like to compute the cost, in hashes, of generating and verifying a signature at any given depth d.

To generate the signature, one needs to do a one-time signature of the message (requiring n hashes). Assuming the signer can cache the tree, no further computation is required. Verification requires one to perform p hashes to verify the current node's signature of the message, and additional p hashes to verify the parent node's signature of the current node. If the receiver has cached the tree, no further computation is required; otherwise, an additional (d-1)p hashes are required.

In contrast to the single signature case, then, each of a sequence of signatures costs n+2p hashes if receivers cache the signature tree, or n+(d+1)p hashes if they do not. How reasonable is it to cache a signature tree? If we choose SHA as our message digest, we need to sign 160 bits of message, for which we could choose n=165 and p=82. Both signer and verifier must cache, for each node, 3n 160-bit numbers (the signer caches the random numbers R, then verifier caches the anchor values H(R)); this works out to 4950 bytes per node. This is easily supported.

In fact, receivers who cache the tree need only maintain the lowest layer of nodes, so that only about half the nodes already traversed need be kept at any time. They can prune additional information off the tree by removing the message signature vectors after they are exhausted.

5 Constructing the Optimal Tree

In Section 4, we showed how to choose n and p to minimize the total cost of an OTS. Now in this section we will describe how to choose the parameters of a k-ary tree with a depth of d. Note that the tree structure does not need to be binary; we can increase k. On the other hand we should stop somewhere in our tree; that is at some point the cost of using our large tree to verify an OTS is more than that of starting a new tree in which we have signed the root node by conventional means. In other words, we need to optimize the value of d.

We will try to minimize the average verification cost of one OTS. Suppose that the verifier only caches the root node of the tree. This is a reasonable assumption, especially when the signer uses the tree to sign messages for different receivers. Suppose also that the cost of verifying a traditional signature is C times more than verifying a OTS. In a k-ary tree with a depth of d, the number of messages that can be signed and the number of OTS required are

$$\sum_{y=1}^{d} k^{y-1}, \sum_{y=1}^{d} yk^{y-1}$$

respectively. In a single tree, the cost per message is

$$\frac{C + \sum_{y=1}^{d} yk^{y-1}}{\sum_{y=1}^{d} k^{y-1}}$$

We try to find the smallest d such that the average cost per message is smaller than it is for d+1. If we use the equality of

$$W = \sum_{y=1}^{d} yk^{y-1} = \frac{1 - k^d - dk^d + dk^{d+1}}{(k-1)^2}$$

$$\frac{C + \sum_{y=1}^{d+1} yk^{y-1}}{\sum_{y=1}^{d+1} k^{y-1}} \approx \frac{C + \sum_{y=1}^{d} yk^{y-1}}{\sum_{y=1}^{d} k^{y-1}}$$

is approximately equal to

$$C + (d+1)k^d + W \approx k(C+W)$$

If we put W into its place and make necessary manipulations, our formula becomes

$$d \approx \frac{\log\left[(k-1)^2 C + 1\right]}{\log k} - 1 \qquad (6)$$

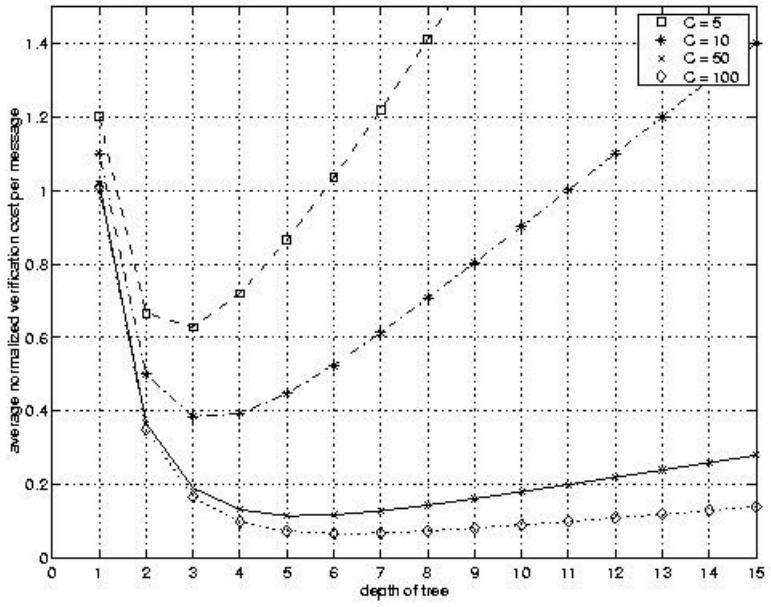

Figure 2: choosing the optimal depth for a binary tree

In Figure 2, we show the depth of a binary tree versus average normalized verification cost per message. One should choose the depth d where the average normalized cost per message is minimal.

6 Conclusion

We have provided a theoretical foundation for evaluating the efficiency and compactness of one-time digital signatures. This work reveals the absolute minimum complexity of computing a digital signature over a space of b-bit messages, based on the weighted costs of signature generation and verification. We have shown how

this theoretical minimum can be achieved, by using a simple and efficient mapping between messages and subsets of random numbers. We have demonstrated how this family of one-time signature schemes fits into an elegant protocol for amortizing the cost of one-time signatures over many messages. Finally, we have provided a theoretical foundation for evaluating the efficiency of the OTS tree structure based on the weighted costs of traditional and one-time signatures.

References

[1] Jurjen N.E. Bos and David Chaum. Provable unforgeable signaturesIn Ernest F. Brickell, editor, Proc. CRYPTO 92}, pages 1-14. Springer-Verlag, 1992. Lecture Notes in Computer Science No. 740.

[2] Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital signatures. In G.Brassard, editor, CRYPTO 89, pages 263--277. Springer-Verlag, 1990. Lecture Notes in Computer Science No.\ 435.

[3] National~Institute for Standards and Technology. Digital Signature Standard (DSS). Federal Register, 56(169), August 30 1991.

[4] L.Lamport. Constructing digital signatures from a one-way function. Technical Report CSL-98, SRI International, October 1979.

[5] A.Menezes, P.Van Oorschot, and S.Vanstone. Handbook of applied cryptography. CRC Press series on discrete mathematics and its applications. CRC Press, 1996. ISBN 0-8493-8523-7.

[6] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, Proc. CRYPTO 87, pages 369--378. Springer-Verlag, 1988. Lecture Notes in Computer Science No. 293.

[7] Ralph C. Merkle. A certified digital signature. In G.Brassard, editor, Proc. CRYPTO 89, pages 218--238. Springer-Verlag, 1990. Lecture Notes in Computer Science No. 435.

[8] R.C. Merkle. Secrecy, authentication, and public key systems, Stanford Univ, Jun 1979.

[9] National Institute of Standards and Technology (NIST). FIPS Publication 180: Secure Hash Standard (SHS), May 11, 1993.

[10] National Institute of Standards and Technology (NIST) FIPS Publication 46-2: Data Encryption Standard, December 30, 1993.

[11] Ronald L. Rivest. The {MD5} message-digest algorithm. April 1992. RFC 1321.

[12] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2):120--126, 1978.

[13] E. Sperner. Ein satz uber untermenge einer endlichen menge. Math Z., 27, pages 544--548, 1928.

Weak Forward Security in Mediated RSA

Gene Tsudik[1]

Department of Information and Computer Science, University of California, Irvine.
Email: gts@ics.uci.edu

Abstract. Mediated RSA (mRSA) [1] is a simple and practical method of splitting RSA private keys between the user and the Security Mediator (SEM). Neither the user nor the SEM can cheat each other since a signature or a decryption must involve both parties. mRSA allows fast and fine-grained control (revocation) of users' security priviliges.
Forward security is an important and desirable feature for signature schemes. Despite some notable recent results, no forward-secure RSA variant has been developed. In this paper (abstract), we show how weak forward security can be efficiently obtained with mediated RSA. We consider several methods, based on both multiplicative and additive mRSA and discuss their respective merits.

1 Forward Security

Forward security is a timely and active research topic which has received some attention in the recent research literature. The purpose of forward security is to mitigate an important problem in ordinary public key signatures: the inability to preserve the validity of past signatures following a compromise of one's private key. In other words, if a forward-secure signature scheme is employed, an adversary who discovers the private key of a user is unable to forge user's signatures from earlier times (pre-dating the compromise).

The notion of forward security was introduced by Anderson [2]. Since then, a number of schemes were proposed. Some are generic, i.e., applicable to any signature scheme [3], while others target (and modify) a particular signature scheme to achieve forward security [4, 5].

In this paper we concentrate on weak forward security in a mediated signature setting. Informally, weak forward security means that the adversary is unable to forge past signatures if she compromises only one (of the two) share-holders of the private key. Specifically, we propose, discuss and analyze two simple schemes built on top of mediated RSA (mRSA), a 2-out-of-2 threshold RSA scheme.

The paper is organized as follows: the next section provides an overview of mRSA. Then, Section 3 describes the forward secure additive mRSA and discusses its security and efficiency features. Section 4 presents another scheme based on multiplicative mRSA. This scheme is more flexible but slightly less efficient. The paper ends with the summary and some directions for future work.

2 Mediated RSA

Mediated RSA (mRSA) involves a special entity, called a SEM (SEcurity Mediator) which is an on-line semi-trusted server. To sign or decrypt a message, Alice must first obtain a message-specific token from the SEM. Without this token Alice can not use her private key. To revoke Alice's ability to sign or decrypt, the administrator instructs the SEM to stop issuing tokens for Alice's public key. At that instant, Alice's signature and/or decryption capabilities are revoked. For scalability reasons, a single SEM serves many users. One of the mRSA's advantages is its transparency: SEM's presence is invisible to other users: in signature mode, mRSA yields standard RSA signatures, while in decryption mode, mRSA accepts plain RSA-encrypted messages.

The main idea behind mRSA is the splitting of an RSA private key into two parts as in threshold RSA [6]. One part is given to a user while the other is given to a SEM. If the user and the SEM cooperate, they employ their respective half-keys in a way that is functionally equivalent to (and indistinguishable from) standard RSA. The fact that the private key is not held in its entirety by any one party is transparent to the outside, i.e., to the those who use the corresponding public key. Also, knowledge of a half-key cannot be used to derive the entire private key. Therefore, neither the user nor the SEM can decrypt or sign a message without mutual consent.

We now provide an overview of mRSA functions. The variant described below is the additive mRSA (+mRSA) as presented by Boneh, et al. in [1]. There is also a multiplicative mRSA variant – denoted *mRSA – where the private key is computed as the product of the two shares. (See Appendix for the description). Muliplicative mRSA was first introduced in the Yaksha system [7] and later discussed in [8].

Algorithm +mRSA.key (executed by CA)

Let k (even) be the security parameter

1. Generate random $k/2$-bit primes: p, q
2. $n \leftarrow pq$
3. $e \xleftarrow{r} Z^*_{\phi(n)}$
4. $d \leftarrow 1/e \bmod \phi(n)$
5. $d_u \xleftarrow{r} Z_n - \{0\}$
6. $d_{sem} \leftarrow (d - d_u) \bmod \phi(n)$
7. $SK \leftarrow d$
8. $PK \leftarrow (n, e)$

After computing the above values, the CA securely communicates d_{sem} to the SEM and d_u – to the user. (A detailed description of this procedure can be found in [1].) The user's public key PK is released, as usual, in a public key certificate.

Protocol +mRSA.sign (executed by User and SEM)

1. USER: $h \leftarrow H(m)$
   where $H()$ is a suitable hash function (e.g., SHA-based HMAC) and $|H()| < k$.
2. USER: send h to SEM.
3. In parallel:
   3.1 SEM:
   (a) If USER revoked return (ERROR)
   (b) $PS_{sem} \leftarrow h^{d_{sem}} \bmod n$
   (c) send PS_{sem} to USER
   3.2 USER:
   (a) $PS_u \leftarrow h^{d_u} \bmod n$
4. USER: $h' \leftarrow (PS_{sem} * PS_u)^e \bmod n$
5. USER: If $h' \neq h$ then return (ERROR)
6. $S \leftarrow (PS_{sem} * PS_u) \bmod n$
7. USER: return (h,S)

The signature verification (+mRSA.ver) algorithm is not provided as it is identical to that in plain RSA.

3 Forward Secure +mRSA

The main idea in forward-secure additive mRSA (FS+mRSA) is for both SEM and user to evolve their private key shares in parallel. The evolution is very simple: each party logically multiplies its share by e. We say "logically" since no actual muliplication is performed; instead, each party merely maintains a counter (i) which is the index of the current time period. As in all other forward secure schemes, there is a maximum number T past which the shares are not evolved.

At any given time, the current private key is:

$$d_i = d_0 * e^i \text{ and } d_0 = d * e^{-T}$$

The user's and SEM's respective key shares, at a given interval are:

$$d_{i,u} = (d_{0,u}) * e^i \text{ where } d_{0,u} = d_u * e^{-T} \bmod \phi(n)$$

and:

$$d^{i,sem} = (d_{0,sem}) * e^i \text{ where } d_{0,sem} = d_{sem} * e^{-T} \bmod \phi(n)$$

The i-th private key evolution can be thus rewritten as:

$$d_i = (d_{0,u}) * e^i + (d_{0,sem}) * e^i = d_0 * e^i = d * e^{i-T}$$

In actuality, both user and SEM always maintain $d_{0,u}$ and $d_{0,sem}$, which are their respective initial shares. However, when they apply their respective shares (to sign a message) they use the current evolution. The reason for not actually

computing $d_{i,u/sem}$ is because neither the SEM nor the user knows $\phi(n)$ and thus cannot compute values of the form:

$$((d_{0,u/sem}) * e^i) \mod \phi(n)$$

Recall that p, q and, consequently, $\phi(n)$ are known only to the CA.

The flavor of forward security offered by our approach is *weak*. Here 'weak' means that, throughout the lifetime of the public key (T periods), the adversary is allowed to compromise only one of the parties' secrets, i.e., only $d_{i,u}$ or $d_{i,sem}$ but not both.

Although the above may be viewed as a drawback, we claim that weak forward security is appropriate for the mRSA setting, since the security of mRSA is based on the non-compromise of both key shares. More specifically, the SEM is an entity more physically secure and more trusted than a regular user. Hence, it makes sense to consider what it takes for mRSA to be forward secure primarily with respect to the user's private key share.

3.1 FS+mRSA in detail

Like most forward-secure signature methods, FS+mRSA is composed of the following four algorithms: `FS+mRSA.key, FS+mRSA.sign FS+mRSA.ver and FS+mRSA.update`. The purpose of the first three is obvious, while `FS+mRSA.update` is the secret key share update algorithm executed by both user and SEM. We do not specify `FS+mRSA.update` since it is trivial: as mentioned above, it does not actually evolve each party's key share: it merely increments the interval counter.

`Algorithm FS+mRSA.key (executed by CA)`

`Let` (t, T) `be the length of the update interval and the max. number of update intervals, respectively.`

1-7. Identical to +mRSA.key
8. $PK \leftarrow (t, T, n, e)$
9. $d_{0,u} \leftarrow d_u * e^{-T} \mod \phi(n)$
10. $d_{0,sem} \leftarrow d_{sem} * e^{-T} \mod \phi(n)$

Protocol FS+mRSA.sign (i,m)

i (0 $\leq$ i $\leq$ T) is the current interval index and m is the input message

1. USER: $h \leftarrow H_i(m)$
   where $H_i()$ is a suitable hash function (e.g., SHA-based HMAC) indexed with the current interval. ($|H_i()| < k$)
2. USER: send m to SEM.
3. In parallel:
   3.1. SEM:
   (a) If USER revoked return (ERROR)
   (b) $h \leftarrow H_i(m)$
   (c) $PS_{sem} \leftarrow (h^{d_{0,sem}})^{e^i} \bmod n$
   (d) send PS_{sem} to USER
   3.2. USER:
   (a) $PS_u \leftarrow (h^{d_{0,u}})^{e^i} \bmod n$
4. USER: $h' \leftarrow (PS_{sem} * PS_u)^{e*e^{T-i}} \bmod n$
5. USER: If $h' \neq h$ then return (ERROR)
6. $S \leftarrow (PS_{sem} * PS_u) \bmod n$
7. USER: return (h,S)

We note that, in steps 3.1.a, 3.2.b and 4, two exponentiations are performed.

Algorithm FS+mRSA.ver (i,S,m,e,n)

i ($0 \leq i \leq T$) is the claimed interval index,
S is the purported signature on a message m, and (e, n) is the public key of the signer

1. if ($i < 0$) or ($i > T$) return (ERROR)
2. $h \leftarrow H_i(m)$
3. $h' \leftarrow S^{e*e^{T-i}}$
4. If $h' \neq h$ then return (0)
5. return (1)

From the descriptions of FS+mRSA.sign and FS+mRSA.ver it is clear that the present scheme is correct, i.e., signature verification succeeds iff a valid signature is provided:

$$S \leftarrow h^{d^i} = h^{d_{i,u}+d_{i,sem}} = h^{d_u*e^{-T}*e^i+d_{sem}*e^{-T}*e^i} = h^{d*e^{i-T}}$$

and

$$S^{e*e^{T-i}} = (h^{d*e^{i-T}})^{e*e^{T-i}} = h^{ed} = h$$

3.2 Efficiency

In [1], the efficiency of mRSA is shown to be roughly equivalent to unoptimized RSA, i.e., RSA without using the Chinese Remainder Theorem (CRT). The

efficiency of FS+mRSA is only slightly lower than that of mRSA. The only difference in signing is the extra exponentiation with e^i performed by both the user and the SEM in parallel.

In general, an additional exponentiation with e^i is also needed for verifying FS+mRSA signatures. However, we observe that, if the user's public exponent is small (e.g., 3), the curent public key $e_i = 3^{i+1}$ is likely to be smaller than the modulus n for many values of i. For example, if $e = 3$ and $k = 1024$, $|e_i| < k$ and $e_i < n$ for $0 \leq i \leq 592$. In that case, e_i can be stored as a single $k - bit$ number and only one exponentiation would be required to verify an FS+mRSA signature.

The extra storage due to forward security in FS+mRSA is negligible. Since key shares are only logically evolved, the only extra information maintained by all signers and SEM-s (on top of what is already required by mRSA) is the index of the current time interval.

3.3 Security Considerations

In all security aspects (other than forward security) the proposed scheme is essentially equivalent to plain RSA as argued in [1]. Similarly, the *forward security* property of FS+mRSA is based on the difficulty of computing roots in a composite-order group which is also the foundation of the RSA cryptosystem. While this extended abstract does not contain a proof of this claim, we provide the following informal argument:

> Assume that the adversary compromises the user at an interval j and, as a result, learns $d_{0,u}$.[1] In order to violate forward security, it suffices for the adversary to generate a single new signature of the form (where $i < j$ and $h = H(m)$ for some new message m):
>
> $$S = h^{d_i} = h^{d_0 * e^i} = h^{d_{0,sem} * e^i + d_{0,u} * e^i} = (h^{d_{0,sem} * e^i} * h^{d_{0,u} * e^i})$$
>
> Computing $h^{d_{0,u} * e^i} \pmod n$ is trivial. However, computing $h^{d_{0,sem} * e^i} \pmod n$ seems to require taking e-th (cube) roots (mod$n$) since, in the current interval j ($j > i$), the SEM is using as its key share $(d_{0,sem} * e^j)$ and only "produces" values of the form: $h^{d_{0,sem} * e^j} \bmod n$.

All forward-secure signature schemes proposed thus far rely on the secure deletion of old secret keys. This is not always a realistic assumption, especially when secure (tamper-resistant) hardware is not readily avaialable. In this aspect, FS+mRSA offers an advantage since the user's secret key share is not actually evolved and no deletion of prior secrets is assumed. While the compromise of the user's current key share yields all user's key shares for all prior intervals, no past-dated signatures can be produced since the SEM's key share evolves separately. We also note that this property is symmetric: if a SEM's key share

[1] This is possible because $d_{j,u}$ is never actually computed, but composed, when needed, as $d_{0,u} * e^j$.

is ever compromised, forward security is preserved as long as the user's share remains secure.

There are, however, two types of attacks unique to FS+mRSA. We refer to the first type as a *future-dating attack*. In it, an adversary obtains a valid signature from the user (m, S) under the current public key (e, n, i). He then takes advantage of the private key structure to construct a valid signature (S') on the same message m dated in some future interval j $(i < j \leq T)$. This can be easily done by computing:

$$S' = S^{e^{j-i}} = (h^{(d_u + d_{sem}) * e^{i-T}})^{e^{j-i}} = h^{d * e^{j-T}}$$

We note that this attack does not compromise the forward security property of the signature scheme. However, it does pose a problem for FS+mRSA. Fortunately, there is an easy fix. It involves hashing the index of the current time interval together with the message. This is already specified in the initial step in protocol `FS+mRSA.sign`. (In other words, instead of $h = H(m)$ we can compute $h = H(m, i)$, or, better yet, $h = \mathrm{HMAC}_i(m)$. This essentially rules out the future-dating attack.)

The second attack type is an *oracle attack*. In it, an adversary, masquarading as the user, sends signature requests to the SEM during the time interval i. This is easy to do since the communication channel between the user and the SEM is neither private nor authentic. The adversary collects a number of "half-signatures" of the form (m, PS_{sem}) where:

$$PS^{i,sem} = h^{d_{sem} * e^{i-T}}$$

Suppose that at a later interval j $(i < j \leq T)$, the adversary actually compromises the user's secret $d_{j,u}$. Although the adversary can not compute "new" signatures from prior time intervals, he can use the previously acquired half-signatures to forge signatures from period i:

$$S' = (PS_{i,sem})^{d_u * e^{i-T}} = (h^{d_{sem} * e^{i-T}})^{d_{sem} * e^{i-T}} = h^{d * e^{i-T}}$$

One simple way of coping with the oracle attack is to require the user-SEM communication channel to be *authentic*. This is not much of a burden in practice due to widely-deployed and available tools such as IPSec [9] and SSL [10]. An alternative is to require mRSA-based authentication of the signature request messages flowing from the user to the SEM. This can be accomplished as follows. When sending a signature request to the SEM, the user computes, in addition: $\bar{h} = \bar{H}(h)$ where $\bar{H}() \neq H()$ is a suitable (cryptographically strong) hash function such that $|\bar{H}()| < k$. He then computes:

$$P\bar{S}_u \leftarrow \bar{h}^{d_{0,u}} \bmod n$$

The user sends $P\bar{S}_u$ along with h in the signature request to the SEM. The SEM, before computing its half-signature (PS_{sem}), verifies $P\bar{S}_u$ by computing: $\bar{h}' = \bar{H}(h)$ and comparing:

$$\bar{h}' \text{ and } (P\bar{S}_u{}^{d_{0,sem}})^{e * e^T} \bmod n$$

Since these two values match only if the user originated the signature request, oracle attacks can be thereby avoided.

4 Forward Secure *mRSA

We now construct another forward-secure scheme based on the multiplicative mRSA variant. Only the key generation and signing algorithms are shown; the verification algorithm is identical to that in FS+mRSA.

Algorithm FS*mRSA.key

1-7. Identical to *mRSA.key (see Appendix)
8. $PK \leftarrow (t, T, n, e)$
9. $d_{0,sem} \leftarrow d_{sem} * e^{-T} \bmod \phi(n)$

The main difference with respect to $FS + mRSA$ is the unilateral update feature. In the present scheme, only the SEM's share is evolved whereas the user's share remains the same throughout the lifetime of the key. This is a desireable feature since it saves the user one exponentiation over FS+mRSA. However, this does not significantly influence the overall performance since, unlike FS+mRSA, the two parties cannot compute their half-signatures in parallel.

Protocol FS*mRSA.sign

1. USER: $h \leftarrow H_i(m)$
2. USER: send m to SEM
3. SEM: If USER revoked return (ERROR)
4. SEM: $h \leftarrow H_i(m)$
5. SEM: $PS_{sem} \leftarrow h^{d_{i,sem}} \bmod n$
6. SEM: send PS_{sem} to USER
7. USER: $h' \leftarrow (PS_{sem}^{d_u})^{e*e^{T-i}} \bmod n$
8. USER: If $h' \neq h$ then return (ERROR)
9. $S \leftarrow (PS_{sem}^{d_u}) \bmod n$
10. USER: return (h,S)

The correctness of FS*mRSA is evident from the verification procedure:

$$(S)^{e*e^{T-i}} = ((PS_{sem})^{d_u})^{e*e^{T-i}} = ((h^{d_{i,sem}})^{d_u})^{e*e^{T-i}} =$$
$$((h^{d_{sem}*e^{i-T}})^{d_u})^{e*e^{T-i}} = (h^{d*e^{i-T}})^{e*e^{T-i}} = h$$

Just like FS+mRSA, this scheme is vulnerable to both future-dating and oracle attacks. Fortunately, the exact countermeasures described in Section 3.3 are equally applicable here.

Another trivial variation of this scheme entails only the user (but not the SEM) evolving its key share. This is a less attractive option since it is much more likely that the user, rather than the SEM, succumbs to eventual compromise. Finally, it is also possible to have both parties evolving the key (just as in FS+mRSA). The main difference here would be that signature verification would require an extra exponentiation with e^{2i} rather than e^i.

5 A Final Observation

A crucial (but purposely ignored above) detail in the construction of `FS+mRSA` and `FS*mRSA` is the use of the current period index i in the hashing of the input message. The intended purpose of the index is as a hedge against possible attacks against the key evolution scheme. However, a closer look indicates that the inclusion of the index in the hash is **in and of itself** sufficient to provide the same weak forward security that we hope to attain with key evolution. In this case, key evolution as described above can be dropped completely to be replaced by the simple hashing of the period index. (This would also result in a much more efficient scheme.)

6 Summary

We described two methods of obtaining efficient (yet weak) forward security with mediated RSA. These methods work with both multiplicative and additive mRSA variants. The degree of forward security is weak since we assume that only the user or the SEM (but not both) are compromised by the adversary. However, this assumption is in line with the mRSA notion of security which is based on the inability to compromise both parties.

Since, aside from signatures, mRSA can be used for encryption, the natural issue to consider is whether FS+mRSA and FS*mRSA schemes are useful for *forward-secure encryption*.

7 Acknowledgements

Many thanks to Giuseppe Ateniese for pointing out an attack on the previous version of `FS+mRSA` as well to anonymous referees for their comments.

References

1. D. Boneh, X. Ding, G. Tsudik, and B. Wong, "Instanteneous revocation of security capabilities," in *Proceeding of USENIX Security Symposium 2001*, Aug. 2001.
2. R. Anderson, "Invited lecture at the acm conference on computer and communication security (ccs'97)," 1997.
3. H. Krawczyk, "Simple forward-secure signatures from any signature scheme," in *ACM Conference on Computer and Communication Security (CCS'00)*, 2000.
4. G. Itkis and L. Reyzin, "Forward-secure signatures with optimal signing and verifying," in *CRYPTO'01*, 2001.
5. M. Bellare and S. Miner, "A forward-secure digital signature scheme," in *CRYPTO'99*, 1999.
6. P. Gemmel, "An introduction to threshold cryptography," *RSA CryptoBytes*, vol. 2, no. 7, 1997.
7. R. Ganesan, "Augmenting kerberos with pubic-key cryptography," in *Symposium on Network and Distributed Systems Security* (T. Mayfield, ed.), (San Diego, California), Internet Society, Feb. 1995.

8. P. MacKenzie and M. K. Reiter, "Networked cryptographic devices resilient to capture," in *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pp. 12–25, May 2001.
9. S. Kent and R. Atkinson, "RFC 2401: Security architecture for the internet protocol," Nov 1998.
10. "The openssl project web page," *http://www.openssl.org*.

A Multiplicative mRSA – *mRSA

The *mRSA variant is largely similar to its additive counterpart. The only substantive difference has to do with parallelizing private key operations by the user and the SEM. In +mRSA, both parties can perform exponentiations with their respective private key shares in parallel. In contrast, *mRSA prescribes serial operation.

Algorithm *mRSA.key (executed by CA)

Let k (even) be the security parameter

1. Generate random $k/2$-bit primes: p, q
2. $n \leftarrow pq$
3. $e \stackrel{r}{\leftarrow} Z^*_{\phi(n)}$
4. $d \leftarrow 1/e \bmod \phi(n)$
5. $d_u \stackrel{r}{\leftarrow} Z^*_{\phi(n)}$
6. $d_{sem} \leftarrow (d/d_u) \bmod \phi(n)$
7. $SK \leftarrow (d, n)$
8. $PK \leftarrow (n, e)$

As in additive mRSA, CA securely communicates d_{sem} to the SEM and d_u – to the user. The user's public key PK is released as part of a public key certificate.

Protocol *mRSA.sign (executed by User and SEM)

1. USER: $h \leftarrow H(m)$
2. USER: send h to SEM
3. SEM: If USER revoked return (ERROR)
4. SEM: $PS_{sem} \leftarrow h^{d_{sem}} \bmod n$
5. SEM: send PS_{sem} to USER
6. USER: $h' \leftarrow (PS_{sem}^{d_u})^e \bmod n$
7. USER: If $h' \neq h$ then return (ERROR)
8. USER: $S \leftarrow (PS_{sem}^{d_u}) \bmod n$
9. USER: return (h,S)

Short Signatures from the Weil Pairing

Dan Boneh
dabo@cs.stanford.edu

Ben Lynn
blynn@cs.stanford.edu

Hovav Shacham
hovav@cs.stanford.edu

Abstract

We introduce a short signature scheme based on the Computational Diffie-Hellman assumption on certain elliptic and hyper-elliptic curves. For standard security parameters, the signature length is about half that of a DSA signature with a similar level of security. Our short signature scheme is designed for systems where signatures are typed in by a human or are sent over a low-bandwidth channel. We survey a number of properties of our signature scheme such as signature aggregation and batch verification.

1 Introduction

Short digital signatures are needed in environments with strong bandwidth constraints. For example, product registration systems often ask users to key in a signature provided on a CD label. When a human is asked to type in a digital signature, the shortest possible signature is needed. Similarly, due to space constraints, short signatures are needed when one prints a bar-coded digital signature on a postage stamp [47, 42]. As a third example, consider legacy protocols that allocate a fixed short field for non-repudiation [1, 29]. One would like to use the most secure signature that fits in the alloted field length.

The two most frequently used signatures schemes, RSA and DSA, produce relatively long signatures compared to the security they provide. For example, when one uses a 1024-bit modulus, RSA signatures are 1024 bits long. Similarly, when one uses a 1024-bit modulus, standard DSA signatures are 320 bits long. Elliptic curve variants of DSA, such as ECDSA, are also 320 bits long [2]. A 320-bit signature is too long to be keyed in by a human.

We propose a signature scheme whose length is approximately 170 bits and which provides a level of security similar to that of 320-bit DSA signatures. Our signature scheme is secure against existential forgery under a chosen-message attack (in the random oracle model), assuming the Computational Diffie-Hellman problem (CDH) is hard on certain elliptic curves over a finite field. Generating a signature is a simple multiplication on the curve. Verifying the signature is done using a bilinear pairing on the curve. Our signature scheme inherently uses properties of curves. Consequently, there is no equivalent of our scheme in $\mathbb{F}_q^*$, the multiplicative group of a finite field.

Constructing short signatures is an old problem. Several proposals show how to shorten DSA while preserving the same level of security. Naccache and Stern [42] propose a variant of DSA where the signature length is approximately 240 bits. Mironov [40] suggests a DSA variant with a similar length and gives a concrete security analysis of the construction in the random oracle model. Another technique proposed for reducing DSA signature length is signatures with message recovery [43, 47]. In such systems one encodes a part of the message into the signature thus

shortening the total length of the message-signature pair. For long messages, one can then achieve a DSA signature overhead of 160 bits. However, for very short messages (e.g., 64 bits) the total length remains 320 bits. Using our signature scheme, the signature length is always on the order of 160 bits, however short the message. We also note that Patarin *et al.* [45, 18] construct short signatures whose security depends on the Hidden Field Equation problem.

Our signature scheme uses groups where the CDH problem is hard, but the Decision Diffie-Hellman problem (DDH) is easy. The first example of such groups was given in [31] and was used in [30, 10]. We call such groups Gap Diffie-Hellman groups, or GDH groups for short. We show how to construct a signature scheme from GDH groups, prove security of the scheme, and show how to build GDH groups that lead to short signatures. The signature scheme resembles the undeniable signature scheme of Chaum and Pedersen [14]. Our signature scheme has several useful properties, described in Section 5. For example, signatures generated by different people on different messages can be aggregated into a single signature [11]. The signature also supports standard extensions such as threshold signatures and blind signatures [9].

Notation. We use $E/\mathbb{F}_q$ to denote an elliptic curve with coefficients in $\mathbb{F}_q$. For $r \geq 1$, we use $E(\mathbb{F}_{q^r})$ to denote the group of points on E in $\mathbb{F}_{q^r}$. We use $|E(\mathbb{F}_{q^r})|$ to denote the number of points in $E(\mathbb{F}_{q^r})$.

2 Gap Diffie-Hellman groups and bilinear maps

Before presenting the signature scheme, we first review a few concepts related to bilinear maps and Gap Diffie-Hellman groups. We use the following notation:

- G_1 and G_2 are two (multiplicative) cyclic groups of prime order p;
- g_1 is a generator of G_1 and g_2 is a generator of G_2;
- ψ is an isomorphism from G_2 to G_1, with $\psi(g_2) = g_1$; and
- e is a bilinear map $e : G_1 \times G_2 \to G_T$.

The group G_T is described below. One can set $G_1 = G_2$, but we allow for the more general case where $G_1 \neq G_2$ so that we can take advantage of certain families of non-supersingular elliptic curves as described in Section 4.3.

The proofs of security require an efficiently computable isomorphism $\psi : G_2 \to G_1$. When $G_1 = G_2$ and $g_1 = g_2$ one could take ψ to be the identity map. When $G_1 \neq G_2$ we will need to describe explicitly an efficiently computable isomorphism $\psi : G_2 \to G_1$. The map ψ is essential for security. To illustrate this, we give in the next section an example of a bilinear map that engenders an insecure signature scheme precisely because ψ does not exist.

With this setup we obtain natural generalizations of the CDH and DDH problems:

Computational co-Diffie-Hellman (co-CDH) on (G_1, G_2)**:** Given $g_2, g_2^a \in G_2$ and $h \in G_1$ as input, compute $h^a \in G_1$.

Decision co-Diffie-Hellman (co-DDH) on (G_1, G_2)**:** Given $g_2, g_2^a \in G_2$ and $h, h^b \in G_1$ as input, output `yes` if $a = b$ and `no` otherwise. When the answer is `yes` we say that (g_2, g_2^a, h, h^a) is a co-Diffie-Hellman tuple.

When $G_1 = G_2$ these problems reduce to standard CDH and DDH.

Next we a define a co-GDH group pair to be a pair of groups (G_1, G_2) on which co-DDH is easy but co-CDH is hard. We define the success probability of an algorithm $\mathcal{A}$ in solving the Computational co-Diffie-Hellman problem on (G_1, G_2) as

$$\mathsf{Succ\ co\text{-}CDH}_{\mathcal{A}} \stackrel{\text{def}}{=} \Pr\left[\mathcal{A}(g_2, g_2^a, h) = h^a : a \stackrel{\text{R}}{\leftarrow} \mathbb{Z}_p, h \stackrel{\text{R}}{\leftarrow} G_1\right] \ .$$

The probability is over the uniform random choice of a from $\mathbb{Z}_p$ and h from G_1, and over the coin tosses of $\mathcal{A}$. We say that an algorithm $\mathcal{A}$ (t, ϵ)-breaks Computational co-Diffie-Hellman on (G_1, G_2) if $\mathcal{A}$ runs in time at most t, and $\mathsf{Succ\ co\text{-}CDH}_{\mathcal{A}}$ is at least ϵ. Here time is measured according to some fixed computational model — say, state transitions in a probabilistic (oracle) Turing machine.

Definition 2.1. Two groups (G_1, G_2) are a (τ, t, ϵ)-Gap co-Diffie-Hellman group pair (co-GDH group pair) if they satisfy the following properties:

- The group operation on both G_1 and G_2 and the map ψ from G_2 to G_1 can be computed in time at most τ.
- The Decision co-Diffie-Hellman problem on (G_1, G_2) can be solved in time at most τ.
- No algorithm (t, ϵ)-breaks Computational co-Diffie-Hellman on (G_1, G_2).

When (G_1, G_1) is a (τ, t, ϵ) co-GDH group pair we say G_1 is a (τ, t, ϵ)-Gap-Diffie-Hellman group (GDH group).

Informally, we are only interested in co-GDH group pairs where τ is sufficiently small so that the co-DDH problem has an efficient solution, but t/ϵ is sufficiently large so that the co-CDH problem is intractable. Currently, the only examples of such Gap Diffie-Hellman groups arise from bilinear maps [31]. We briefly define bilinear maps and show how they give GDH groups. It is possible that other constructions for useful Gap Diffie-Hellman groups exist.

Let G_1 and G_2 be two groups as above, with an additional group G_T such that $|G_1| = |G_2| = |G_T|$. A bilinear map is a map $e : G_1 \times G_2 \to G_T$ with the following properties:

- Bilinear: for all $u \in G_1, v \in G_2$ and $a, b \in \mathbb{Z}$, $e(u^a, v^b) = e(u, v)^{ab}$.
- Non-degenerate: $e(g_1, g_2) \neq 1$.

Definition 2.2. Two order-p groups (G_1, G_2) are a (τ, t, ϵ)-bilinear group pair if they satisfy the following properties:

- The group operation on both G_1 and G_2 and the map ψ from G_2 to G_1 can be computed in time at most τ.
- A group G_T of order p and a bilinear map $e : G_1 \times G_2 \to G_T$ exist, and e is computable in time at most τ.
- No algorithm (t, ϵ)-breaks Computational co-Diffie-Hellman on (G_1, G_2).

Joux and Nguyen [31] showed that an efficiently-computable bilinear map e provides an algorithm for solving the Decision co-Diffie-Hellman problem as follows: For a tuple (g_2, g_2^a, h, h^b) where $h \in G_1$ we have

$$a = b \bmod p \quad \Longleftrightarrow \quad e(h, g_2^a) = e(h^b, g_2) \ .$$

Consequently, if two groups (G_1, G_2) are a (τ, t, ϵ)-bilinear group pair, then they are also a $(2\tau, t, \epsilon)$-co-GDH group pair. The converse is probably not true.

3 Signature schemes based on Gap Diffie-Hellman groups

We present a signature scheme that works on any Gap co-Diffie-Hellman group pair (G_1, G_2). We prove security of the scheme and, in the next section, show how it leads to short signatures. The scheme resembles the undeniable signature scheme proposed by Chaum and Pedersen [14]. Okamoto and Pointcheval [44] briefly note that gap problems can give rise to signature schemes. However, most gap problems will not lead to short signatures.

Let (G_1, G_2) be (t, ϵ)-Gap co-Diffie-Hellman group pair where $|G_1| = |G_2| = p$. A signature σ is an element of G_1. The signature scheme comprises three algorithms, *KeyGen*, *Sign*, and *Verify*. It makes use of a full-domain hash function $H : \{0,1\}^* \to G_1$. The security analysis views H as a random oracle [7]. In Section 3.2 we weaken the requirement on the hash function H.

Key generation. Pick random $x \stackrel{\mathrm{R}}{\leftarrow} \mathbb{Z}_p$ and compute $v \leftarrow g_2^x$. The public key is $v \in G_2$. The private key is x.

Signing. Given a private key $x \in \mathbb{Z}_p$, and a message $M \in \{0,1\}^*$, compute $h \leftarrow H(M) \in G_1$ and $\sigma \leftarrow h^x$. The signature is $\sigma \in G_1$.

Verification. Given a public key $v \in G_2$, a message $M \in \{0,1\}^*$, and a signature $\sigma \in G_1$, compute $h \leftarrow H(M) \in G_1$ and verify that (g_2, v, h, σ) is a valid co-Diffie-Hellman tuple. If so, output `valid`; if not, output `invalid`.

A signature is a single element of G_1. To construct short signatures, therefore, we need co-GDH group pairs where elements in G_1 have a short representation. We construct such groups in Section 4.

3.1 Security

We prove the security of the signature scheme against existential forgery under adaptive chosen-message attacks in the random oracle model. Existential unforgeability under a chosen message attack [28] for a signature scheme (*KeyGen*, *Sign*, and *Verify*) is defined using the following game between a challenger and an adversary $\mathcal{A}$:

Setup. The challenger runs algorithm *KeyGen* to obtain a public key PK and private key SK. The adversary $\mathcal{A}$ is given PK.

Queries. Proceeding adaptively, $\mathcal{A}$ requests signatures with PK on at most q_S messages of his choice $M_1, \ldots, M_{q_s} \in \{0,1\}^*$. The challenger responds to each query with a signature $\sigma_i = \mathit{Sign}(SK, M_i)$.

Output. Eventually, $\mathcal{A}$ outputs a pair (M, σ) and wins the game if (1) M is not any of $M_1, \ldots, M_{q_s}$, and (2) $\mathit{Verify}(PK, M, \sigma) = \texttt{valid}$.

We define $\mathsf{Adv}\,\mathsf{Sig}_{\mathcal{A}}$ to be the probability that $\mathcal{A}$ wins in the above game, taken over the coin tosses of *KeyGen* and of $\mathcal{A}$.

Definition 3.1. A forger $\mathcal{A}$ (t, q_S, q_H, ϵ)-breaks a signature scheme if $\mathcal{A}$ runs in time at most t, $\mathcal{A}$ makes at most q_S signature queries and at most q_H queries to the hash function, and $\mathsf{Adv}\,\mathsf{Sig}_{\mathcal{A}}$ is at least ϵ. A signature scheme is (t, q_S, q_H, ϵ)-existentially unforgeable under an adaptive chosen-message attack if no forger (t, q_S, q_H, ϵ)-breaks it.

The following theorem shows that the signature scheme is secure. Security of the scheme follows from the hardness of co-CDH on (G_1, G_2). When $G_1 = G_2$ security is based on the standard Computational Diffie-Hellman assumption in G_1.

Theorem 3.2. *Let (G_1, G_2) be a (τ, t', ϵ')-co-GDH group pair of order p. Then the signature scheme on (G_1, G_2) is (t, q_S, q_H, ϵ)-secure against existential forgery under an adaptive chosen-message attack (in the random oracle model), for all t and ϵ satisfying*

$$\epsilon \geq e(q_S + 1) \cdot \epsilon' \qquad \text{and} \qquad t \leq t' - c_{G_1}(q_H + 2q_S) \ .$$

Here c_{G_1} is a constant that depends on G_1, and e is the base of the natural logarithm.

Proof. Suppose $\mathcal{A}$ is a forger algorithm that (t, q_S, q_H, ϵ)-breaks the signature scheme. We show how to construct a t'-time algorithm $\mathcal{B}$ that solves co-CDH on (G_1, G_2) with probability at least ϵ'. This will contradict the fact that (G_1, G_2) is a (t', ϵ')-co-GDH group pair.

Let g_2 be a generator of G_2. Algorithm $\mathcal{B}$ is given $g_2, u \in G_2$ and $h \in G_1$, where $u = g_2^a$. Its goal is to output $h^a \in G_1$. Algorithm $\mathcal{B}$ simulates the challenger and interacts with forger $\mathcal{A}$ as follows.

Setup. Algorithm $\mathcal{B}$ starts by giving $\mathcal{A}$ the generator g_2 and the public key $u \cdot g_2^r \in G_2$, where r is random in $\mathbb{Z}_p$.

H-queries. At any time algorithm $\mathcal{A}$ can query the random oracle H. To respond to these queries algorithm $\mathcal{B}$ maintains a list of tuples $\langle M_j, w_j, b_j, c_j \rangle$ as explained below. We refer to this list as the H-list. The list is initially empty. When $\mathcal{A}$ queries the oracle H at a point $M_i \in \{0,1\}^*$, algorithm $\mathcal{B}$ responds as follows:

1. If the query M_i already appears on the H-list in a tuple $\langle M_i, w_i, b_i, c_i \rangle$ then algorithm $\mathcal{B}$ responds with $H(M_i) = w_i \in G_1$.
2. Otherwise, $\mathcal{B}$ generates a random coin $c_i \in \{0,1\}$ so that $\Pr[c_i = 0] = 1/(q_S + 1)$.
3. Algorithm $\mathcal{B}$ picks a random $b_i \in \mathbb{Z}_p$ and computes $w_i \leftarrow h^{1-c_i} \cdot \psi(g_2)^{b_i} \in G_1$.
4. Algorithm $\mathcal{B}$ adds the tuple $\langle M_i, w_i, b_i, c_i \rangle$ to the H-list and responds to $\mathcal{A}$ by setting $H(M_i) = w_i$.

Note that either way w_i is uniform in G_1 and is independent of $\mathcal{A}$'s current view as required.

Signature queries. Let M_i be a signature query issued by $\mathcal{A}$. Algorithm $\mathcal{B}$ responds to this query as follows:

1. Algorithm $\mathcal{B}$ runs the above algorithm for responding to H-queries to obtain a $w_i \in G_1$ such that $H(M_i) = w_i$. Let $\langle M_i, w_i, b_i, c_i \rangle$ be the corresponding tuple on the H-list. If $c_i = 0$ then $\mathcal{B}$ reports failure and terminates.
2. Otherwise, we know $c_i = 1$ and hence $w_i = \psi(g_2)^{b_i} \in G_1$. Define $\sigma_i = \psi(u)^{b_i} \cdot \psi(g_2)^{r b_i} \in G_1$. Observe that $\sigma_i = w_i^{a+r}$ and therefore σ_i is a valid signature on M_i under the public key $u \cdot g_2^r = g_2^{a+r}$. Algorithm $\mathcal{B}$ gives σ_i to algorithm $\mathcal{A}$.

Output. Eventually algorithm $\mathcal{A}$ produces a message-signature pair (M_f, σ_f) such that no signature query was issued for M_f. If there is no tuple on the H-list containing M_f then $\mathcal{B}$ issues a query itself for $H(M_f)$ to ensure that such a tuple exists. We assume σ_f is a valid signature on M_f under the given public key; if it is not, $\mathcal{B}$ reports failure and terminates. Next, algorithm $\mathcal{B}$

finds the tuple $\langle M_f, w, b, c\rangle$ on the H-list. If $c = 1$ then $\mathcal{B}$ reports failure and terminates. Otherwise, $c = 0$ and therefore $H(M_f) = w = h \cdot \psi(g_2)^b$. Hence, $\sigma = h^{a+r} \cdot \psi(g_2)^{b(a+r)}$. Then $\mathcal{B}$ outputs the required h^a as $h^a \leftarrow \sigma/(h^r \cdot \psi(u)^b \cdot \psi(g_2)^{rb})$.

This completes the description of algorithm $\mathcal{B}$. It remains to show that $\mathcal{B}$ solves the given instance of the co-CDH problem on (G_1, G_2) with probability at least ϵ'. To do so, we analyze the three events needed for $\mathcal{B}$ to succeed:

$\mathcal{E}_1$**:** $\mathcal{B}$ does not abort as a result of any of $\mathcal{A}$'s signature queries.

$\mathcal{E}_2$**:** $\mathcal{A}$ generates a valid message-signature forgery (M_f, σ_f).

$\mathcal{E}_3$**:** Event $\mathcal{E}_2$ occurs and $c = 0$ for the tuple containing M_f on the H-list.

$\mathcal{B}$ succeeds if all of these events happen. The probability $\Pr[\mathcal{E}_1 \wedge \mathcal{E}_3]$ is:

$$\Pr[\mathcal{E}_1 \wedge \mathcal{E}_3] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \Pr[\mathcal{E}_3 \mid \mathcal{E}_1 \wedge \mathcal{E}_2] \ . \tag{1}$$

The following claims give a lower bound for each of these terms.

Claim 1. *The probability that algorithm $\mathcal{B}$ does not abort as a result of $\mathcal{A}$'s signature queries is at least $1/e$. Hence, $\Pr[\mathcal{E}_1] \geq 1/e$.*

Proof. Without loss of generality we assume that $\mathcal{A}$ does not ask for the signature of the same message twice. We prove by induction that after $\mathcal{A}$ makes i signature queries the probability that $\mathcal{B}$ does not abort is at least $(1 - 1/(q_S + 1))^i$. The claim is trivially true for $i = 0$. Let M_i be $\mathcal{A}$'s i'th signature query and let $\langle M_i, w_i, b_i, c_i\rangle$ be the corresponding tuple on the H-list. Then prior to issuing the query, the bit c_i is independent of $\mathcal{A}$'s view — the only value that could be given to $\mathcal{A}$ that depends on c_i is $H(M_i)$, but the distribution on $H(M_i)$ is the same whether $c_i = 0$ or $c_i = 1$. Therefore, the probability that this query causes $\mathcal{B}$ to abort is at most $1/(q_S + 1)$. Using the inductive hypothesis and the independence of c_i, the probability that $\mathcal{B}$ does not abort after this query is at least $(1 - 1/(q_S + 1))^i$. This proves the inductive claim. Since $\mathcal{A}$ makes at most q_S signature queries the probability that $\mathcal{B}$ does not abort as a result of all the signature queries is at least $(1 - 1/(q_S + 1))^{q_S} \geq 1/e$. □

Claim 2. *If algorithm $\mathcal{B}$ does not abort as a result of $\mathcal{A}$'s signature queries then algorithm $\mathcal{A}$'s view is identical to its view in the real attack. Hence, $\Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \geq \epsilon$.*

Proof. The public key given to $\mathcal{A}$ is from the same distribution as a public key produced by algorithm *KeyGen*. Responses to H-queries are as in the real attack since each response is uniformly and independently distributed in G_1. All responses to signature queries are valid. Therefore, $\mathcal{A}$ will produce a valid message-signature pair with probability at least ϵ. Hence, $\Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \geq \epsilon$. □

Claim 3. *The probability that algorithm $\mathcal{B}$ does not abort after $\mathcal{A}$ outputs a valid forgery is at least $1/(q_S + 1)$. Hence, $\Pr[\mathcal{E}_3 \mid \mathcal{E}_1 \wedge \mathcal{E}_2] = 1/(q_S + 1)$.*

Proof. Given that events $\mathcal{E}_1$ and $\mathcal{E}_2$ happened, algorithm $\mathcal{B}$ will abort only if $\mathcal{A}$ generates a forgery (M_f, σ_f) for which the tuple $\langle M_f, w, b, c\rangle$ on the H-list has $c = 1$. At the time $\mathcal{A}$ generates its output it knows the value of c_i for those M_i for which it issued a signature query. All the remaining c_i's are independent of $\mathcal{A}$'s view. Indeed, if $\mathcal{A}$ did not issue a signature query for M_i then the only value given to $\mathcal{A}$ that depends on c_i is $H(M_i)$, but the distribution on $H(M_i)$ is the same whether $c_i = 0$ or $c_i = 1$. Since $\mathcal{A}$ could not have issued a signature query for M_f we know that c is independent of $\mathcal{A}$'s current view and therefore $\Pr[c = 0 \mid \mathcal{E}_1 \wedge \mathcal{E}_2] = 1/(q_S + 1)$ as required. □

Using the bounds from the claims above in equation (1) shows that $\mathcal{B}$ produces the correct answer with probability at least $\epsilon/e(q_S+1) \geq \epsilon'$ as required. Algorithm $\mathcal{B}$'s running time is the same as $\mathcal{A}$'s running time plus the time it takes to respond to (q_H+q_S) hash queries and q_S signature queries. Each query requires an exponentiation in G_1 which we assume takes time c_{G_1}. Hence, the total running time is at most $t + c_{G_1}(q_H + 2q_S) \leq t'$ as required. This completes the proof of Theorem 3.2. □

The analysis used in the proof of Theorem 3.2 resembles Coron's analysis of the Full Domain Hash (FDH) signature scheme [16]. We note that the security analysis can be made tight using Probabilistic Full Domain Hash (PFDH) [17], at the cost of increasing signature length. The security reduction in Theorem 3.2 can also be made tight without increasing signature length via the technique of Katz and Wang [32].

Our signature scheme requires an algorithm for deciding DDH. In groups where a DDH-deciding algorithm is not available, Goh and Jarecki [27] show that it is still possible to construct a signature scheme based on CDH, at the cost of substantially greater signature length.

The necessity of $\psi : G_2 \to G_1$. Recall that the proof of security relied on the existence of an efficiently computable isomorphism $\psi : G_2 \to G_1$. To show the necessity of ψ we give an example of a bilinear map $e : G_1 \times G_2 \to G_T$ for which the co-CDH problem is believed to be hard on (G_1, G_2) and yet the resulting signature scheme is insecure.

Let q be a prime and let G_2 be a subgroup of $\mathbb{Z}_q^*$ of prime order p with generator g. Let G_1 be the group $G_1 = \mathbb{Z}_p$ with addition. Define the map $e : G_1 \times G_2 \to G_2$ as $e(x, y) = y^x$. The map is clearly bilinear since $e(ax, y^b) = e(x, y)^{ab}$. The co-CDH problem on (G_1, G_2) is as follows: Given $g, g^a \in G_2$ and $x \in G_1$ compute $ax \in G_1$. The problem is believed to be hard since an algorithm for computing co-CDH on (G_1, G_2) gives an algorithm for computing discrete log in G_2. Hence, (G_1, G_2) satisfies all the conditions of Theorem 3.2 except that there is no known computable isomorphism $\psi : G_2 \to G_1$. It is is easy to see that the resulting signature scheme from this bilinear map is insecure. Given one message-signature pair, it is easy to recover the private key.

We comment that one can avoid using ψ at the cost of making a stronger complexity assumption. Without ψ the necessary assumption for proving security is that no polynomial time algorithm can compute $h^a \in G_1$ given $g_2, g_2^a \in G_2$ and $g_1, g_1^a, h \in G_1$. Since ψ naturally exists in all the group pairs (G_1, G_2) we are considering, there is no reason to rely on this stronger complexity assumption.

3.2 Hashing onto elliptic curves

The signature scheme needs a hash function $H : \{0,1\}^* \to G_1$. In the next section we use elliptic curves to construct co-GDH group pairs and therefore we need a hash function $H : \{0,1\}^* \to G_1$ where G_1 is a subgroup of an elliptic curve. Since it is difficult to build hash functions that hash directly onto a subgroup of an elliptic curve we slightly relax the hashing requirement.

Let $\mathbb{F}_q$ be a field of characteristic greater than 2. Let $E/\mathbb{F}_q$ be an elliptic curve defined by $y^2 = f(x)$ and let $E(\mathbb{F}_q)$ have order m. Let $P \in E(F_q)$ be a point of prime order p, where p^2 does not divide m. We wish to hash onto the subgroup $G_1 = \langle P \rangle$. Suppose we are given a hash function $H' : \{0,1\}^* \to \mathbb{F}_q \times \{0,1\}$. Such hash functions H' can be built from standard cryptographic hash functions. The security analysis will view H' as a random oracle. We use the following deterministic algorithm called *MapToGroup* to hash messages in $\{0,1\}^*$ onto G_1. Fix a small parameter $I = \lceil \log_2 \log_2(1/\delta) \rceil$, where δ is some desired bound on the failure probability.

***MapToGroup*$_{H'}$:** The algorithm defines $H : \{0,1\}^* \to G_1$ as follows:

1. Given $M \in \{0,1\}^*$, set $i \leftarrow 0$;

2. Set $(x, b) \leftarrow H'(i \parallel M) \in \mathbb{F}_q \times \{0,1\}$, where i is represented as an I-bit string;

3. If $f(x)$ is a quadratic residue in $\mathbb{F}_q$ then do:

   3a. Let $y_0, y_1 \in \mathbb{F}_q$ be the two square roots of $f(x)$. We use $b \in \{0,1\}$ to choose between these roots. Choose some full ordering of $\mathbb{F}_q$ and ensure that y_1 is greater than y_0 according to this ordering (swapping y_0 and y_1 if necessary). Set $\tilde{P}_M \in E(\mathbb{F}_q)$ to be the point $\tilde{P}_M = (x, y_b)$.

   3b. Compute $P_M = (m/p)\tilde{P}_M$. Then P_M is in G_1. If $P_M \neq \mathcal{O}$, output $MapToGroup_{H'}(M) = P_M$ and stop; otherwise, continue with Step 4.

4. Otherwise, increment i, and go to Step 2; if i reaches 2^I, report failure.

The failure probability can be made arbitrarily small by picking an appropriately large I. For each i, the probability that $H'(i \parallel M)$ leads to a point on G_1 is approximately $1/2$ (where the probability is over the choice of the random oracle H'). Hence, the expected number of calls to H' is approximately 2, and the probability that a given message M will be found unhashable is $1/2^{(2^I)} \leq \delta$.

Lemma 3.3. *Let $E/\mathbb{F}_q$ be an elliptic curve and let $E(\mathbb{F}_q)$ have order m. Let G_1 be a subgroup of $E(\mathbb{F}_q)$ of order p such that p^2 does not divide m. Suppose the co-GDH signature scheme is (t, q_S, q_H, ϵ)-secure in the groups (G_1, G_2) when a random hash function $H : \{0,1\}^* \to G_1$ is used. Then it is $(t - 2^I c_{G_1} q_H,\ q_H - q_S - 1,\ q_S,\ \epsilon)$-secure when the hash function H is computed with $MapToGroup_{H'}$ and H' is a random oracle hash function $H' : \{0,1\}^* \to \mathbb{F}_q \times \{0,1\}$. Here c_{G_1} is a constant that depends on G_1.*

Proof. Suppose a forger algorithm $\mathcal{F}'$ (t, q_H, q_S, ϵ)-breaks the co-GDH signature scheme on (G_1, G_2) when given access to a random oracle $H' : \{0,1\}^* \to \mathbb{F}_q \times \{0,1\}$ and $MapToGroup_{H'}$. We build an algorithm $\mathcal{F}$ that $(t + 2^I c_{G_1}(q_H + q_S + 1), q_H + q_S + 1, q_S, \epsilon)$-breaks the co-GDH signature scheme when given access to a full-domain random oracle hash $H : \{0,1\}^* \to G_1$.

Setup. To respond to queries made by $\mathcal{F}'$, $\mathcal{F}$ uses an array s_{ij}, whose entries are elements of $\mathbb{F}_q \times \{0,1\}$. The array has q_H rows and 2^I columns. On initialization, $\mathcal{F}$ fills s_{ij} with uniformly-selected elements of $\mathbb{F}_q \times \{0,1\}$.

Algorithm $\mathcal{F}$ has access to a random oracle $H : \{0,1\}^* \to G_1$. It will use this to simulate the random oracle $H' : \{0,1\}^* \to \mathbb{F}_q \times \{0,1\}$ that $\mathcal{F}'$ uses. Algorithm $\mathcal{F}$ is also given a public key v, and a signing oracle for that key. Its goal is to output a forgery on some message under v.

Algorithm $\mathcal{F}$ runs $\mathcal{F}'$ and responds to its oracle queries as follows.

H'-queries. Algorithm $\mathcal{F}$ keeps track (and indexes) all the unique messages M_i for which $\mathcal{F}'$ requests an H' hash.

When $\mathcal{F}'$ asks for an H' hash of a message $w \parallel M_i$ whose M_i the forger $\mathcal{F}$ had not previously seen (and whose w is an arbitrary I-bit string), $\mathcal{F}$ must fix up row i of its matrix s before responding.

It scans the row s_{ij}, $0 \leq j < 2^I$. For each entry $s_{ij} = (x, b)$, $\mathcal{F}$ follows Step 3 of *MapToGroup*, above, seeking points in $G_1 \setminus \{\mathcal{O}\}$. If none of the entries s_{ij} yields a point in $G_1 \setminus \{\mathcal{O}\}$, row s_{ij},

$0 \le j < 2^I$, is not patched. Otherwise, for the smallest j for which s_{ij} maps into $G_1 \setminus \{\mathcal{O}\}$, $\mathcal{F}$ replaces s_{ij} with a different point (x_i, b_i) defined as follows. Let $Q_i = H(M_i) \in G_1$. If $Q_i \in G_1 \setminus \mathcal{O}$, then $\mathcal{F}$ constructs a random point $\tilde{Q}_i \in E(\mathbb{F}_q)$ satisfying $(m/p)\tilde{Q}_i = Q_i$ as follows:

1. Let $z = (m/p)^{-1} \bmod p$. Note that m/p is integer since p divides m. Furthermore, m/p has an inverse modulo p since p^2 does not divide m and hence m/p is relatively prime to p.
2. Pick a random point $T_i \in E(\mathbb{F}_q)$.
3. Set $\tilde{Q}_i = (x_i, y_i) = pT_i + zQ_i$.

Then $\tilde{Q}_i$ is a random point in $E(\mathbb{F}_q)$ such that $(m/p)\tilde{Q}_i = Q_i$. Algorithm $\mathcal{F}$ sets $s_{ij} = (x_i, b_i)$ where $b_i \in \{0, 1\}$ is set so that (x_i, b_i) maps to $\tilde{Q}_i$ in Step 3(a) of *MapToGroup*. Note that $MapToGroup_{H'}(M_i)$ now equals $H(M_i)$, and that the distribution of s_{ij} is not changed by the patching.

If $Q_i = \mathcal{O}$, then $xQ_i = Q_i = \mathcal{O}$ for all $x \in \mathbb{Z}_p$, and in particular for the private key x corresponding to the challenge public key v. Algorithm $\mathcal{F}$ outputs the forgery $(M_i, \mathcal{O})$ and halts. This forgery is nontrivial because $\mathcal{F}$ always queries its H oracle at a message M_i before querying its signing oracle at M_i.

Once this preliminary patching has been completed, $\mathcal{F}$ is able to answer H' hash queries by $\mathcal{F}'$ for strings $w \parallel M_i$ by simply returning s_{iw}. The simulated H' which $\mathcal{F}'$ sees is perfectly indistinguishable from that in the real attack.

Signature queries. Algorithm $\mathcal{F}$ is asked for a signature on some message M_i, indexed as above. It first runs its H' algorithm above to fix up the row corresponding to M_i in its s matrix. This computation queries the H oracle at M_i and may cause $\mathcal{F}$ to abort, having discovered a trivial forgery. If the computation does not abort, $MapToGroup_{H'}(M_i) = H(M_i)$ holds. Algorithm $\mathcal{F}$ queries its own signing oracle at M_i, obtaining a signature $\sigma_i \in G_1$, which is also the correct signature under the $MapToGroup_{H'}$ hash function. Algorithm $\mathcal{F}$ responds to the query with σ_i.

Output. Finally, $\mathcal{F}'$ halts. It either concedes failure, in which case so does $\mathcal{F}$, or it returns a message M^* and a nontrivial forged signature σ^*. Algorithm $\mathcal{F}'$ must not have queried its signing oracle at M^*, so neither did $\mathcal{F}$.

Algorithm $\mathcal{F}$ first runs its H' algorithm above to fix up the row corresponding to M^* in its s matrix. This assigns to M^* an index i^*, such that $M_{i^*} = M^*$. This computation queries the H oracle at M_{i^*} and may cause $\mathcal{F}$ to abort, having discovered a trivial forgery.

If the computation does not abort, $MapToGroup_{H'}(M^*) = H(M^*)$ holds. Thus σ^* is a valid forgery on message M^* under hash function H as well as $MapToGroup_{H'}$. Since $\mathcal{F}$ did not query its signing oracle at M^*, the forgery is nontrivial for it as well as for $\mathcal{F}'$. Algorithm $\mathcal{F}$ outputs the valid and nontrivial forgery (M^*, σ^*) and halts.

Algorithm $\mathcal{F}$ succeeds if either it discovers a trivial forgery (a message M such that $H(M) = \mathcal{O}$), or it perfectly simulates the environment that $\mathcal{F}'$ expects. Whenever $\mathcal{F}'$ succeeds in creating a nontrivial forgery, so does $\mathcal{F}$. If $\mathcal{F}'$ succeeds with probability ϵ, so does $\mathcal{F}$. If $\mathcal{F}'$ takes time t to run, $\mathcal{F}$ takes time t, plus the s-array fix-up time that is potentially necessary on each hash query, each signing query, and at the final output phase. If running the exponentiation in Step 3

of the *MapToGroup* algorithm takes time c_{G_1}, then $\mathcal{F}$ takes time at most $t + 2^I c_{G_1}(q_H + q_S + 1)$. Algorithm $\mathcal{F}$ potentially makes a hash query for each hash query made by $\mathcal{F}'$, for each signing query made by $\mathcal{F}'$, and in the final output phase. Algorithm $\mathcal{F}$ makes a signing query for each signing query made by $\mathcal{F}'$.

Thus if $\mathcal{F}'$ (t, q_H, q_S, ϵ)-breaks the co-GDH signature scheme when given access to a random oracle $H' : \{0,1\}^* \to \mathbb{F}_q \times \{0,1\}$ and $MapToGroup_{H'}$, then $\mathcal{F}$ $(t+2^I c_{G_1}(q_H+q_S+1), q_H+q_S+1, q_S, \epsilon)$-breaks the co-GDH signature scheme when given access to a full-domain random oracle hash $H : \{0,1\}^* \to G_1$.

Conversely, if the co-GDH signature scheme is (t, q_H, q_S, ϵ)-secure when instantiated with a full-domain random oracle hash $H : \{0,1\}^* \to G_1$, it is $(t - 2^I c_{G_1} q_H, q_H - q_S - 1, q_S, \epsilon)$ when instantiated with a random oracle $H' : \{0,1\}^* \to \mathbb{F}_q \times \{0,1\}$ and $MapToGroup_{H'}$. □

4 Building co-GDH group pairs with small representations

Using the Weil [34, pp. 243–245] and Tate [21] pairings, we obtain co-GDH group pairs from certain elliptic curves. We recall some necessary facts about elliptic curves (see, e.g., [34, 50]), and then show how to use certain curves for short signatures.

4.1 Elliptic curves and the Weil pairing

Our goal is to construct bilinear groups (G_1, G_2) which lead to co-GDH group pairs as discussed in Section 2. Let $E/\mathbb{F}_q$ be an elliptic curve. We first define a useful constant called the security multiplier of a subgroup $\langle P \rangle \subseteq E(\mathbb{F}_q)$.

Definition 4.1. Let q be a prime power, and $E/\mathbb{F}_q$ an elliptic curve with m points in $E(\mathbb{F}_q)$. Let P in $E(\mathbb{F}_q)$ be a point of prime order p where $p^2 \nmid m$. We say that the subgroup $\langle P \rangle$ has a security multiplier α, for some integer $\alpha > 0$, if the order of q in $\mathbb{F}_p^*$ is α. In other words:

$$p \mid q^\alpha - 1 \quad \text{and} \quad p \nmid q^k - 1 \quad \text{for all } k = 1, 2, \ldots, \alpha - 1 \ .$$

The security multiplier of $E(\mathbb{F}_q)$ is the security multiplier of the largest prime order subgroup in $E(\mathbb{F}_q)$.

We describe two families of curves that provide $\alpha = 6$. For standard security parameters this is sufficient for obtaining short signatures. It is an open problem to build useful elliptic curves with slightly higher α, say $\alpha = 10$ (see Section 4.5).

Our first step is to define G_1 and G_2. We will then describe a bilinear map $e : G_1 \times G_2 \to G_T$, describe an isomorphism $\psi : G_2 \to G_1$, and discuss the intractability of co-CDH on (G_1, G_2).

Balasubramanian-Koblitz. Let $E/\mathbb{F}_q$ be an elliptic curve and let $P \in E(\mathbb{F}_q)$ be a point of prime order p with $p \nmid q$. Suppose the subgroup $\langle P \rangle$ has security multiplier $\alpha > 1$, i.e. $p \nmid q - 1$. Then, a useful result of Balasubramanian and Koblitz [3] shows that $E(\mathbb{F}_{q^\alpha})$ contains a point Q of order p that is linearly independent of P. We set $G_1 = \langle P \rangle$ and $G_2 = \langle Q \rangle$. Then $|G_1| = |G_2| = p$. Note that $G_1 \subseteq E(\mathbb{F}_q)$ and $G_2 \subseteq E(\mathbb{F}_{q^\alpha})$.

The Weil and Tate pairings. With notation as above, let $E[p]$ be the group of points of order dividing p in $E(F_{q^\alpha})$. Then the group $E[p]$ is isomorphic to $\mathbb{Z}_p \times \mathbb{Z}_p$ [50] and $G_1, G_2 \leq E[p]$. The Weil pairing is a map $e : E[p] \times E[p] \to \mathbb{F}_{q^\alpha}^*$ with the following properties:

(i) Identity: for all $R \in E[p]$, $e(R, R) = 1$.

(ii) Bilinear: for all $R_1, R_2 \in E[p]$ and $a, b \in \mathbb{Z}$ we have $e(aR_1, bR_2) = e(R_1, R_2)^{ab}$.

(iii) Non-degenerate: if for $R \in E[p]$ we have $e(R, R') = 1$ for all $R' \in E[p]$, then $R = \mathcal{O}$. It follows that $e(P, Q) \neq 1$.

(iv) Computable: for all $R_1, R_2 \in E[p]$, the pairing $e(R_1, R_2)$ can be computed in polynomial time [39].

Note that $e(R_1, R_2) = 1$ if and only if R_1 and R_2 are linearly dependent. See [37, 10] for a definition of the Weil pairing and a description of the algorithm for computing it. The Tate pairing [21] is another useful bilinear map on $E[p]$. It has properties similar to those of the Weil pairing, but does not necessarily satisfy Property (i) (identity).

The Weil pairing on the curve E induces a computable, non-degenerate bilinear map $e : G_1 \times G_2 \to \mathbb{F}^*_{q^\alpha}$ which enables us to solve the Decision co-Diffie-Hellman problem on the group pair (G_1, G_2). When the Tate pairing induces a non-degenerate map on $G_1 \times G_2$, it can also be used to solve Decision co-Diffie-Hellman on (G_1, G_2).

The trace map. We present a computable isomorphism $\psi : G_2 \to G_1$, using the trace map, tr, which sends points in $E(\mathbb{F}_{q^\alpha})$ to $E(\mathbb{F}_q)$. Let $\sigma_1, \ldots, \sigma_\alpha$ be the Galois maps of $\mathbb{F}_{q^\alpha}$ over $\mathbb{F}_q$. Also, for $R = (x, y) \in E(\mathbb{F}_{q^\alpha})$ define $\sigma_i(R) = (\sigma_i(x), \sigma_i(y))$. Then the trace map $\mathrm{tr} : E(\mathbb{F}_{q^\alpha}) \to E(\mathbb{F}_q)$ is defined by:

$$\mathrm{tr}(R) = \sigma_1(R) + \cdots + \sigma_\alpha(R) \ .$$

Proposition 4.2. *Let $P \in E(\mathbb{F}_q)$ be a point of prime order $p \neq q$ and let $\langle P \rangle$ have security multiplier $\alpha > 1$. Let $Q \in E(\mathbb{F}_{q^\alpha})$ be a point of order p that is linearly independent of P. If $\mathrm{tr}(Q) \neq \mathcal{O}$ then* tr *is an isomorphism from $\langle Q \rangle$ to $\langle P \rangle$.*

Proof. Suppose $R \in E(\mathbb{F}_q)$ is a point of order p. If R is not in $\langle P \rangle$ then P and R generate $E[p]$ and therefore $E[p] \subseteq E(\mathbb{F}_q)$. It follows that $e(P, R) \in \mathbb{F}^*_q$ has order p since otherwise e would be degenerate on $E[p]$. But since $\alpha > 1$ we know that p does not divide $q - 1$ and consequently there are no elements of order p in $\mathbb{F}^*_q$. Hence, we must have $R \in \langle P \rangle$. It follows that all the points in $E(\mathbb{F}_q)$ of order p are contained in $\langle P \rangle$. Since $\mathrm{tr}(Q) \neq \mathcal{O}$, we know that $\mathrm{tr}(Q) \in E(\mathbb{F}_q)$ has order p and therefore $\mathrm{tr}(Q) \in \langle P \rangle$. Hence, tr is an isomorphism from $\langle Q \rangle$ to $\langle P \rangle$. □

Hence, when $\mathrm{tr}(Q) \neq \mathcal{O}$, the trace map is an isomorphism from G_2 to G_1 and is computable in polynomial time in α and $\log q$ as required.

Intractability of co-CDH on (G_1, G_2). The remaining question is the difficulty of the co-CDH problem on (G_1, G_2). We review necessary conditions for CDH intractability. The best known algorithm for solving co-CDH on (G_1, G_2) is to compute discrete-log in G_1. In fact, the discrete-log and CDH problems in G_1 are known to be computationally equivalent given some extra information about the group G_1 [35]. Therefore, it suffices to consider necessary conditions for making the discrete-log problem on $E(\mathbb{F}_q)$ intractable.

Let $\langle P \rangle$ be a subgroup of $E(\mathbb{F}_q)$ of order p with security multiplier α. We briefly discuss two standard ways for computing discrete-log in $\langle P \rangle$.

1. **MOV:** Use an efficiently computable homomorphism, as in the MOV reduction [36], to map the discrete log problem in $\langle P \rangle$ to a discrete log problem in some extension of $\mathbb{F}_q$, say $\mathbb{F}_{q^i}$. We then solve the discrete log problem in $\mathbb{F}_{q^i}^*$ using the Number Field Sieve algorithm [49]. The image of $\langle P \rangle$ under this homomorphism must be a subgroup of $\mathbb{F}_{q^i}^*$ of order p. Thus we have $p|(q^i - 1)$, which by the definition of α implies that $i \geq \alpha$. Hence, the MOV method can, at best, reduce the discrete log problem in $\langle P \rangle$ to a discrete log problem in a subgroup of $\mathbb{F}_{q^\alpha}^*$. Therefore, to ensure that discrete log is hard in $\langle P \rangle$ we want curves where α is sufficiently large to make discrete log in $\mathbb{F}_{q^\alpha}^*$ intractable.

2. **Generic:** Generic discrete log algorithms such as Baby-Step-Giant-Step and Pollard's Rho method [38] have a running time proportional to $\sqrt{p} \log p$. Therefore, we must ensure that p is sufficiently large.

In summary, we want curves $E/\mathbb{F}_q$ where both a generic discrete log algorithm in $E(\mathbb{F}_q)$ and the Number Field Sieve in $\mathbb{F}_{q^\alpha}^*$ are intractable. At the same time, since our signature scheme has signatures of length $\lceil \log_2 q \rceil$ and public keys of length $\lceil \alpha \log_2 q \rceil$, we wish to keep q as small as possible.

4.2 Co-GDH signatures from elliptic curves

We summarize the construction for co-GDH group pairs and adapt the signature scheme to use a group of points on an elliptic curve.
The co-GDH group pair (G_1, G_2) we use is defined as follows:

1. Let $E/\mathbb{F}_q$ be an elliptic curve and let $P \in E(\mathbb{F}_q)$ be a point of prime order p where $p \nmid q(q-1)$ and $p^2 \nmid |E(\mathbb{F}_q)|$.

2. Let $\alpha > 1$ be the security multiplier of $\langle P \rangle$. We assume $\alpha < p$. By Balasubramanian and Koblitz [3] there exists a point $Q \in E(\mathbb{F}_{q^\alpha})$ that is linearly independent of P. It is easy to construct such a Q in expected polynomial time once the number of points in $E(\mathbb{F}_{q^\alpha})$ is known. Since $\alpha > 1$ we know that $Q \notin E(\mathbb{F}_q)$. We ensure that $\mathrm{tr}(Q) \neq \mathcal{O}$. If $\mathrm{tr}(Q) = \mathcal{O}$ replace Q by $Q + P$. Then $Q + P$ is of order p, it is linearly independent of P, and $\mathrm{tr}(Q + P) \neq \mathcal{O}$ since $\mathrm{tr}(P) = \alpha P \neq \mathcal{O}$.

3. Set $G_1 = \langle P \rangle$ and $G_2 = \langle Q \rangle$.

4. Since P and Q are linearly independent, the Weil pairing gives a non-degenerate bilinear map $e : G_1 \times G_2 \to \mathbb{F}_{q^\alpha}^*$. It can be computed in polynomial time in α and $\log q$. When the Tate pairing is non-degenerate on $G_1 \times G_2$ it can also be used as a bilinear map.

5. Since $\mathrm{tr}(Q) \neq \mathcal{O}$ the trace map on $E(\mathbb{F}_{q^\alpha})$ is an isomorphism from G_2 to G_1 computable in polynomial time in α and $\log q$.

With these subgroups G_1, G_2 of the elliptic curve $E/\mathbb{F}_q$ the signature scheme works as follows. Recall that $\mathit{MapToGroup}_{H'}$ is a hash function $\mathit{MapToGroup}_{H'} : \{0,1\}^* \to G_1$ built from a hash function $H' : \{0,1\}^* \to \mathbb{F}_q^* \times \{0,1\}$ as described in Section 3.2.

Key generation. Pick random $x \xleftarrow{R} \mathbb{Z}_p$, and compute $V \leftarrow xQ$. The public key is $V \in E(\mathbb{F}_{q^\alpha})$. The private key is x.

Signing. Given a private key $x \in \mathbb{Z}_p$, and a message $M \in \{0,1\}^*$, do:

1. Compute $R \leftarrow \mathit{MapToGroup}_{H'}(M) \in G_1$,
2. $\sigma \leftarrow xR \in E(\mathbb{F}_q)$, and
3. output the x-coordinate of σ as the signature s on M. Then $s \in \mathbb{F}_q$.

Verification. Given a public key $V \in G_2$, a message $M \in \{0,1\}^*$, and a signature $s \in \mathbb{F}_q$ do:

1. Find a $y \in \mathbb{F}_q$ such that $\sigma = (s, y)$ is a point in $E(\mathbb{F}_q)$. If no such y exists, output `invalid` and stop.
2. Ensure that σ has order p. If it does not, output `invalid` and stop.
3. Compute $R \leftarrow \mathit{MapToGroup}_{H'}(M) \in G_1$,
4. Test if either $e(\sigma, Q) = e(R, V)$ or $e(\sigma, Q)^{-1} = e(R, V)$.
   If so, output `valid`; Otherwise, output `invalid`.

The signature length is $\lceil \log_2 q \rceil$. Note that during verification we accept the signature if $e(\sigma, Q)^{-1} = e(R, V)$. This is to account for the fact that the signature $s \in \mathbb{F}_q$ could have come from either the point σ or $-\sigma$ in $E(\mathbb{F}_q)$.

Security. By Theorem 3.2 it suffices to study the difficulty of co-CDH on (G_1, G_2). The best known algorithm for solving the co-CDH problem on (G_1, G_2) requires the computation of a discrete log in G_1 or the computation of a discrete log in $\mathbb{F}^*_{q^\alpha}$.

4.3 Using non-supersingular curves over fields of high characteristic

It remains to build elliptic curves with the desired security multiplier α. In the next two sections we show curves with security multiplier, $\alpha = 6$. We begin by describing a family of non-supersingular elliptic curves with $\alpha = 6$. This family is outlined by Miyaji *et al.* [41]. We call these MNT curves.

The idea is as follows: Suppose $q = (2\ell)^2 + 1$ and $p = (2\ell)^2 - 2\ell + 1$ for some $\ell \in \mathbb{Z}$. Then it can be verified that p divides $q^6 - 1$, but does not divide $q^i - 1$ for $0 < i < 6$. So, when p is prime, a curve $E/\mathbb{F}_q$ with p points is likely to have security multiplier $\alpha = 6$.

To build $E/\mathbb{F}_q$ with p points as above we use complex multiplication [8, chapter VIII]. We briefly explain how to do so. Suppose we had integers y, t and another positive integer $D = 3 \bmod 4$ such that

$$q = (t^2 + Dy^2)/4 \tag{2}$$

is an integer prime. Then the complex multiplication method will produce an elliptic curve $E/\mathbb{F}_q$ with $q + 1 - t$ points in time $O(D^2(\log q)^3)$. The value t is called the trace of the curve.

We want a curve over $\mathbb{F}_q$ with p points where $q = (2\ell)^2 + 1$ and $p = (2\ell)^2 - 2\ell + 1$. Therefore, $t = q + 1 - p = 2\ell + 1$. Plugging these values into (2) we get $4((2\ell)^2 + 1) = (2\ell + 1)^2 + Dy^2$ which leads to:

$$(6\ell - 1)^2 - 3Dy^2 = -8 \ . \tag{3}$$

For a fixed $D = 3 \bmod 4$, we need integers ℓ, y satisfying the equation above such that $q = (2\ell)^2 + 1$ is prime and $p = (2\ell)^2 - 2\ell + 1$ is prime (or is a small multiple of a prime). For any such solution we can verify that we get a curve $E(\mathbb{F}_q)$ with security multiplier $\alpha = 6$. Finding integer solutions ℓ, y to an equation of type (3) is done by reducing it to Pell's equation, whose solution is well known [51].

Discriminant D	Signature Size $\lceil \log_2 q \rceil$	DLog Security $\lceil \log_2 p \rceil$	MOV Security $\lceil 6 \log_2 q \rceil$
13368643	149	149	894
254691883	150	147	900
8911723	157	157	942
62003	159	158	954
12574563	161	161	966
1807467	163	163	978
6785843	168	166	1008
28894627	177	177	1062
153855691	185	181	1110
658779	199	194	1194
1060147	203	203	1218
20902979	204	204	1224
9877443	206	206	1236

Table 1: Non-supersingular elliptic curves for co-GDH Signatures. E is a curve over the prime field $\mathbb{F}_q$ and p is the largest prime dividing its order. The MOV reduction maps the curve onto the field $\mathbb{F}_{q^6}$. D is the discriminant of the complex multiplication field of $E/\mathbb{F}_q$.

Table 1 gives some values of D that lead to suitable curves for our signature scheme. For example, we get a curve $E/\mathbb{F}_q$ where q is a 168-bit prime. Signatures using this curve are 168-bits while the best algorithm for co-CDH on $E(\mathbb{F}_q)$ requires either (1) a generic discrete log algorithm taking time approximately 2^{83}, or (2) a discrete log in a 1008-bit finite field of large characteristic.

4.4 A special supersingular curve

Another method for building curves with security multiplier $\alpha = 6$ is to use a special supersingular curve $E/\mathbb{F}_3$. Specifically, we use the curve $E : y^2 = x^3 + 2x \pm 1$ over $\mathbb{F}_3$. The MOV reduction maps the discrete log problem in $E(\mathbb{F}_{3^\ell})$ to $\mathbb{F}^*_{3^{6\ell}}$. We use two simple lemmas to describe the behavior of these curves. (See also [54, 33].)

Lemma 4.3. *The curve E^+ defined by $y^2 = x^3 + 2x + 1$ over $\mathbb{F}_3$ satisfies*

$$|E^+(\mathbb{F}_{3^\ell})| = \begin{cases} 3^\ell + 1 + \sqrt{3 \cdot 3^\ell} & \text{when} \quad \ell = \pm 1 \bmod 12, \text{and} \\ 3^\ell + 1 - \sqrt{3 \cdot 3^\ell} & \text{when} \quad \ell = \pm 5 \bmod 12. \end{cases}$$

The curve E^- defined by $y^2 = x^3 + 2x - 1$ over $\mathbb{F}_3$ satisfies

$$|E^-(\mathbb{F}_{3^\ell})| = \begin{cases} 3^\ell + 1 - \sqrt{3 \cdot 3^\ell} & \text{when} \quad \ell = \pm 1 \bmod 12, \text{and} \\ 3^\ell + 1 + \sqrt{3 \cdot 3^\ell} & \text{when} \quad \ell = \pm 5 \bmod 12. \end{cases}$$

Proof. See [33, Section 2]. □

Lemma 4.4. *Let E be an elliptic curve defined by $y^2 = x^3 + 2x \pm 1$ over $\mathbb{F}_{3^\ell}$, where $\ell \bmod 12$ equals ± 1 or ± 5. Then $|E(\mathbb{F}_{3^\ell})|$ divides $3^{6\ell} - 1$.*

Proof. See [54]. □

Together, Lemmas 4.3 and 4.4 show that, for the relevant values of ℓ, groups on the curves $E^+/\mathbb{F}_{3^\ell}$ and $E^-/\mathbb{F}_{3^\ell}$ will have security multiplier α at most 6 (more specifically: $\alpha \mid 6$). Whether the security parameter actually is 6 for a particular prime subgroup of a curve must be determined by computation.

Automorphism of $E^+, E^-/\mathbb{F}_{3^{6\ell}}$: Both curves E^+ and E^- have a useful automorphism that make the prime-order subgroups of $E^+(\mathbb{F}_{3^\ell})$ and $E^-(\mathbb{F}_{3^\ell})$ into GDH groups (as opposed to co-GDH group pairs). This fact can be used to shrink the size of the public key since it makes it possible for the public key to live in $E(\mathbb{F}_{3^\ell})$ as opposed to $E(\mathbb{F}_{3^{6\ell}})$.

The automorphism is defined as follows. For ℓ such that $\ell \bmod 12$ is ± 1 or ± 5, compute three elements of $\mathbb{F}_{3^{6\ell}}$, u, r^+, and r^-, satisfying $u^2 = -1$, $(r^+)^3 + 2r^+ + 2 = 0$, and $(r^-)^3 + 2r^- - 2 = 0$. Now consider the following maps over $\mathbb{F}_{3^{6\ell}}$:

$$\phi^+(x, y) = (-x + r^+, uy) \quad \text{and} \quad \phi^-(x, y) = (-x + r^-, uy) \ .$$

Lemma 4.5. *Let $\ell \bmod 12$ equal ± 1 or ± 5. Then ϕ^+ is an automorphism of $E^+/\mathbb{F}_{3^{6\ell}}$ and ϕ^- is an automorphism of $E^-/\mathbb{F}_{3^{6\ell}}$. Moreover, if P is a point of order p on $E^+/\mathbb{F}_{3^\ell}$ (or on $E^-/\mathbb{F}_{3^\ell}$) then $\phi^+(P)$ (or $\phi^-(P)$) is a point of order p that is linearly independent of P.*

Proof. See Silverman [50, p. 326, Case II]. □

Let $E/\mathbb{F}_{3^\ell}$ be one of E^+ or E^- and let $P \in E(\mathbb{F}_{3^\ell})$ be a point of prime order p. Set $G_1 = \langle P \rangle$, the group generated by P. Let $\phi : E(\mathbb{F}_{3^\ell}) \to E(\mathbb{F}_{3^{6\ell}})$ be the automorphism of the curve from above. Define the modified Weil pairing $\hat{e} : G_1 \times G_1 \to \mathbb{F}^*_{3^{6\ell}}$ as follows: $\hat{e}(P_1, P_2) = e(P_1, \phi(P_2))$ where e is the standard Weil pairing on $E[p]$. By Lemma 4.5 we know that $\phi(P)$ is linearly independent of P. Therefore, $\hat{e}$ is non-degenerate. It follows that G_1 is a GDH group; ϕ acts as a distortion map [53, 31]. This has two implications for the signature scheme:

- Security of the signature scheme is based on the difficulty of the standard Computational Diffie-Hellman problem in G_1 (as opposed to the co-CDH problem).
- Public keys are elements of G_1 and, hence, are shorter than public keys should the automorphism not exist.

Useful curves. Some useful instantiations of these curves are presented in Table 2. Note that we restrict these instantiations to those where ℓ is prime, to avoid Weil-descent attacks [24, 25], except for $\ell = 121$. It has recently been shown that certain Weil-descent attacks are not effective for this case [19].

Performance. Galbraith *et al.* [23] and Barreto *et al.* [4] show that the Frobenius map on the curves E^+, E^- can be used to speed the computation of the Weil and Tate pairings on these curves. This results in a significant speed-up to the signature-verification algorithm. Consequently, the signature scheme using these curves is much faster than the scheme using the curves from the previous section.

The bad news. MOV reduces the discrete log problem on $E^+(\mathbb{F}_{3^\ell})$ and $E^-(\mathbb{F}_{3^\ell})$ to a discrete log problem in $\mathbb{F}^*_{3^{6\ell}}$. A discrete-log algorithm due to Coppersmith [15, 49] is specifically designed to compute discrete log in small characteristic fields. Consequently, a discrete-log problem in $\mathbb{F}^*_{3^n}$ is much easier than a discrete-log problem in $\mathbb{F}^*_p$ where p is a prime of approximately the same size

curve	l	Sig Size $\lceil \log_2 3^\ell \rceil$	DLog Security $\lceil \log_2 p \rceil$	MOV Security $\lceil 6 \log_2 3^\ell \rceil$
E^-	79	126	126	752
E^+	97	154	151	923
E^+	121	192	155	1151
E^+	149	237	220	1417
E^+	163	259	256	1551
E^-	163	259	259	1551
E^+	167	265	262	1589

Table 2: Supersingular elliptic curves for GDH signatures. Here p is the largest prime divisor of $|E(\mathbb{F}_{3^\ell})|$. The MOV reduction maps the curve onto a field of characteristic 3 of size $3^{6\ell}$.

as 3^n. To get security equivalent to DSA using a 1024-bit prime, we would have to use a curve $E(\mathbb{F}_{3^\ell})$ where $3^{6\ell}$ is much larger than 1024 bits. This leads to much longer signatures, defeating the point of using these curves. In other words, for a fixed signature length, these supersingular curves lead to a signature with reduced security compared to the curves of Section 4.3.

4.5 An open problem: higher security multipliers

With the curves of Section 4.3, a security multiplier of $\alpha = 6$ is sufficient for constructing short signatures with security comparable to DSA using a 1024-bit prime. However, to obtain security comparable to DSA using a 2048-bit prime with $\alpha = 6$ we get signatures of length $2048/6 = 342$ bits. Elliptic curves with higher α, say $\alpha = 10$, would result in short signatures when higher security is needed (such as 2048-bit discrete-log security).

Let q be a large prime power, say, $q > 2^{160}$. It is currently an open problem to construct an elliptic curve $E/\mathbb{F}_q$ such that $E(\mathbb{F}_q)$ has $\alpha = 10$ and $E(\mathbb{F}_q)$ has prime order. Several constructions [5, 20, 13] show how to build elliptic curves E such that $E(\mathbb{F}_q)$ has a given security multiplier α. However, the largest prime order subgroup of $E(\mathbb{F}_q)$ is much smaller than q. For example, the constructions of [5, 20] give curves $E/\mathbb{F}_q$ where the largest prime factor of $|E(\mathbb{F}_q)|$ is of order $\sqrt{q}$. Discrete log in such groups takes time approximately $q^{1/4}$. Therefore, for a given security parameter, the resulting signatures are of the same length as DSA signatures. The constructions in [13] give curves where the largest prime factor of $|E(\mathbb{F}_q)|$ is greater than $\sqrt{q}$, but still substantially smaller than q. These curves result in signatures that are shorter than DSA, but longer than half the size of DSA. The open problem is to build elliptic curves $E/\mathbb{F}_q$ with a given security multiplier α where $E(F_q)$ has prime order. Such curves would provide signatures that are half the size of DSA for any given security level.

One could also build GDH groups of higher genus. Galbraith [22] constructs supersingular curves of higher genus with a "large" security multiplier. For example, the Jacobian of the supersingular curve $y^2 + y = x^5 + x^3$ has security multiplier 12 over $\mathbb{F}_{2^\ell}$. Since a point on the Jacobian of this curve of genus 2 is characterized by two values in $\mathbb{F}_{2^\ell}$ (the two x-coordinates in a reduced divisor), the length of the signature is 2ℓ bits. Hence, we might obtain a signature of length 2ℓ where the security depends on computing CDH in the finite field $\mathbb{F}_{2^{12\ell}}$. This factor of 6 between the length of the signature and the degree of the finite field is the same as in the elliptic curve case. Hence, this genus 2 curve does not improve the security of the signature, but does give more variety in curves used for short signatures. Discrete log on the Jacobian of these curves is reducible to discrete-log in

a field of characteristic 2 and consequently one must take Coppersmith's discrete log algorithm [15] into account, as discussed at the end of Section 4.4.

To obtain larger security multipliers, Rubin and Silverberg [48] propose certain Abelian varieties. They show that signatures produced using the curve of Section 4.4 can be shortened by 20%. The result is an n-bit signature where the pairing reduces the discrete log problem to a finite field of size approximately $2^{7.5n}$. This is the only useful example we currently know of where the multiplier is greater than 6.

5 Extensions

Our signatures support threshold signatures and batch verification. Surprisingly, signatures from distinct people on distinct messages can be aggregated into a single convincing signature. We briefly survey these extensions here and refer to Boldyreva [9], Verheul [52], and Boneh *et al.* [11] for a full description and proofs of security.

5.1 Aggregate signatures

Common environments require managing many signatures by different parties on distinct messages. For example, certificate chains contain signatures on distinct certificates issued by various Certificate Authorities. Our signature scheme enables us to aggregate multiple signatures by distinct entities on distinct messages into a single short signature. Any party that has all the signatures can aggregate signatures, and aggregation can be done incrementally: Two signatures are aggregated, then a third is added to the aggregate, and so on. See [11] for more applications.

Let (G_1, G_2) be a bilinear group pair of prime order p. Suppose n users each have a public-private key pair. For $i = 1, \ldots, n$, user i has private key $x_i \in \mathbb{Z}_p$ and public key $v_i = g_2^{x_i} \in G_2$.

Suppose user i signs a message $M_i \in \{0,1\}^*$ to obtain the signature $\sigma_i = H(M_i)^{x_i} \in G_1$. The aggregate of all these signatures is computed simply as $\sigma \leftarrow \sigma_1 \sigma_2 \cdots \sigma_n \in G_1$.

Aggregate verification: We are given all public keys $v_1, \ldots, v_n \in G_2$, all messages $M_1, \ldots, M_n \in \{0,1\}^*$, and the aggregate signature $\sigma \in G_1$. To verify that, for all $i = 1, \ldots, n$, user i has signed message M_i, we test that

1. The messages $M_1, \ldots, M_n$ are all distinct, and
2. $e(\sigma, g_2) = \prod_{i=1}^{n} e(H(M_i), v_i)$.

If both conditions hold, we accept the aggregate signature. Otherwise, we reject.

We refer to [11] for the exact security model and the proof of security. An attacker who can existentially forge an aggregate signature can be used to solve co-CDH on (G_1, G_2). We note that aggregate signature verification requires a bilinear map — a generic Gap Diffie-Hellman group is apparently insufficient. Generic Gap Diffie-Hellman groups are sufficient for verifying aggregate signatures on the same message by different people, or for verifying aggregate signatures on distinct messages by the same person.

5.2 Batch verification

Suppose n users all sign the same message $M \in \{0,1\}^*$. We obtain n signatures $\sigma_1, \ldots, \sigma_n$. We show that these n signatures can be verified as a batch much faster than verifying them one by one. A similar property holds for other signature schemes [6].

Let (G_1, G_2) be a co-GDH group pair of prime order p. Suppose user i's private key is $x_i \in \mathbb{Z}_p$ and his public key is $v_i = g_2^{x_i} \in G_2$. Signature σ_i is $\sigma_i = H(M)^{x_i} \in G_1$. To verify the n signatures as a batch we use a technique due to Bellare *et al.* [6]:

1. Pick random integers $c_1, \ldots, c_n$ from the range $[0, B]$ for some value B. This B controls the error probability as discussed below.
2. Compute $V \leftarrow \prod_{i=1}^{n} v_i^{c_i} \in G_2$ and $U \leftarrow \prod_{i=1}^{n} \sigma_i^{c_i} \in G_1$.
3. Test that $(g_2, V, H(M), U)$ is a co-Diffie-Hellman tuple. Accept all n signatures if so; reject otherwise.

Theorem 3.3 of [6] shows that we incorrectly accept the n signatures with probability at most $1/B$. Hence, verifying the n signatures as a batch is faster than verifying them one by one. Note that if all signers are required to prove knowledge of their private keys, then taking $c_1 = \cdots = c_n = 1$ is sufficient, yielding even faster batch verification [9]. A similar batch verification procedure can be used to verify quickly n signatures on distinct messages issued by the *same* public key.

5.3 Threshold signatures

Using standard secret sharing techniques [38], our signature scheme gives a robust t-out-of-n threshold signature [9]. In a threshold signature scheme, there are n parties where each possesses a share of a private key. Each party can use its share of the private key to produce a share of a signature on some message M. A complete signature on M can only be constructed if at least t shares of the signature are available.

A robust t-out-of-n threshold signature scheme derives from our signature scheme as follows. A central authority generates a public/private key pair. Let $x \in \mathbb{Z}_p$ be the private key and $v = g_2^x \in G_2$ be the public key. The central authority picks a random polynomial $\omega \in \mathbb{Z}_p[X]$ of degree at most $t-1$ such that $\omega(0) = x$. For $i = 1, \ldots, n$, the authority gives user i the value $x_i = \omega(i)$, its share of the private key. The authority publishes the public key v and n values $u_i = g_2^{x_i} \in G_2$.

When a signature on a message $M \in \{0,1\}^*$ is needed each party that wishes to participate in signature generation publishes its share of the signature as $\sigma_i = H(M)^{x_i} \in G_1$. Without loss of generality, assume users $1, \ldots, t$ participate and generate shares $\sigma_1, \ldots, \sigma_t$. Anyone can verify that share σ_i is valid by checking that $(g_2, u_i, H(M), \sigma_i)$ is a co-Diffie-Hellman tuple. When all t shares are valid, the complete signature is recovered as

$$\sigma \leftarrow \prod_{i=1}^{t} \sigma_i^{\lambda_i} \quad \text{where} \quad \lambda_i = \frac{\prod_{i=1, j \neq i}^{t} (0 - j)}{\prod_{i=1, j \neq i}^{t} (i - j)} \pmod{p} \ .$$

If fewer than t users are able to generate a signature on some message M then these users can be used to solve co-CDH on (G_1, G_2) [9]. This threshold scheme is robust: A participant who contributes a bad partial signature σ_i will be detected immediately since $(g_2, u_i, H(M), \sigma_i)$ will not be a co-Diffie-Hellman tuple.

We note that there is no need for a trusted third party to generate shares of the private key. The n users can generate shares of the private key without the help of a trusted third party using the protocol due to Gennaro *et al.* [26], which is a modification of a protocol due to Pedersen [46]. This protocol does not rely on the difficulty of DDH for security and can thus be employed on Gap Diffie-Hellman groups.

6 Conclusions

We presented a short signature based on bilinear maps on elliptic curves. A signature is only one element in a finite field. Standard signatures based on discrete log such as DSA require two elements. Our signatures are much shorter than all current variants of DSA for the same security. We showed that the scheme is existentially unforgeable under a chosen message attack (in the random oracle model), assuming the Computational Diffie-Hellman problem is hard on certain elliptic-curve groups. More generally, the signature scheme can be instantiated on any Gap Diffie-Hellman group or co-GDH group pair.

We presented two families of elliptic curves that are suitable for obtaining short signatures. The first, based on [41], is a family of non-supersingular curves over a prime finite field. The second uses supersingular curves over $\mathbb{F}_{3^\ell}$. Both families of curves produce n-bit signatures and the discrete log problem on these curves is reducible to a discrete log problem in a finite field of size approximately 2^{6n}. Using the first family of curves, for 1024-bit security we get signatures of size approximately $\lceil 1024/6 \rceil = 171$ bits.

We expect that the first family of curves (the non-supersingular curves) will be the one used for short signatures: 171-bit signatures with 1024-bit security. As discussed at the end of Section 4.4, the second family of curves (the supersingular curve over $\mathbb{F}_{3^\ell}$) should not be used for short signatures. The problem is that discrete log on these curves reduces to a discrete log in a finite field of characteristic 3 where Coppersmith's algorithm can be used.

Implementation results [23, 4] indicate that the signature scheme performs well. Signature generation is just a simple multiplication on an elliptic curve and is faster than RSA signature generation. Verification requires two computations of the bilinear map and is slower than RSA signature verification.

In Section 4.5 we outlined an open problem that would enable us to get even better security while maintaining the same length signatures. We hope future work on constructing elliptic curves or higher genus curves will help in solving this problem.

Acknowledgments

The authors thank Steven Galbraith, Alice Silverberg, Moni Naor, Victor Shoup, Scott Renfro, Paulo Barreto, and Doug Kuhlman for helpful discussions about this work.

References

[1] ANSI X9 Committee. DSTU X9.59-2000: Electronic commerce for the financial services industry: Account-based secure payment objects. `http://www.x9.org/`.

[2] ANSI X9.62 and FIPS 186-2. Elliptic curve digital signature algorithm, 1998.

[3] R. Balasubramanian and N. Koblitz. The improbability that an elliptic curve has subexponential discrete log problem under the Menezes-Okamoto-Vanstone algorithm. *J. Cryptology*, 11(2):141–5, Mar. 1998.

[4] P. Barreto, H. Kim, B. Lynn, and M. Scott. Efficient implementation of pairing-based cryptosystems. *J. Cryptology*, 17(4):321–34, Sept. 2004.

[5] P. Barreto, B. Lynn, and M. Scott. Constructing elliptic curves with prescribed embedding degrees. In S. Cimato, C. Galdi, and G. Persiano, editors, *Proceedings of SCN 2002*, volume 2576 of *LNCS*, pages 257–67. Springer-Verlag, Sept. 2003.

[6] M. Bellare, J. Garay, and T. Rabin. Fast batch verification for modular exponentiation and digital signatures. In K. Nyberg, editor, *Proceedings of Eurocrypt 1998*, volume 1403 of *LNCS*, pages 236–50. Springer-Verlag, May–Jun. 1998.

[7] M. Bellare and P. Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In U. Maurer, editor, *Proceedings of Eurocrypt 1996*, volume 1070 of *LNCS*, pages 399–416. Springer-Verlag, May 1996.

[8] I. F. Blake, G. Seroussi, and N. Smart. *Elliptic Curves in Cryptography*, volume 265 of *London Mathematical Society Lecture Notes*. Cambridge University Press, 1999.

[9] A. Boldyreva. Efficient threshold signature, multisignature and blind signature schemes based on the gap-Diffie-Hellman-group signature scheme. In Y. Desmedt, editor, *Proceedings of PKC 2003*, volume 2567 of *LNCS*, pages 31–46. Springer-Verlag, Jan. 2003.

[10] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. *SIAM J. Computing*, 32(3):586–615, 2003. Extended abstract in *Proceedings of Crypto 2001*.

[11] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In E. Biham, editor, *Proceedings of Eurocrypt 2003*, volume 2656 of *LNCS*, pages 416–32. Springer-Verlag, May 2003.

[12] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In C. Boyd, editor, *Proceedings of Asiacrypt 2001*, volume 2248 of *LNCS*, pages 514–32. Springer-Verlag, Dec. 2001. Full paper: `http://crypto.stanford.edu/~dabo/pubs.html`.

[13] F. Brezing and A. Weng. Elliptic curves suitable for pairing based cryptography. Cryptology ePrint Archive, Report 2003/143, 2003. `http://eprint.iacr.org/`.

[14] D. Chaum and T. Pedersen. Wallet databases with observers. In E. Brickell, editor, *Proceedings of Crypto 1992*, volume 740 of *LNCS*, pages 89–105. Springer-Verlag, Aug. 1992.

[15] D. Coppersmith. Fast evaluation of logarithms in fields of characteristic two. *IEEE Tran. Info. Th.*, 30(4):587–94, 1984.

[16] J.-S. Coron. On the exact security of full domain hash. In M. Bellare, editor, *Proceedings of Crypto 2000*, volume 1880 of *LNCS*, pages 229–35. Springer-Verlag, Aug. 2000.

[17] J.-S. Coron. Optimal security proofs for PSS and other signature schemes. In L. Knudsen, editor, *Proceedings of Eurocrypt 2002*, volume 2332 of *LNCS*, pages 272–87. Springer-Verlag, May 2002.

[18] N. Courtois, M. Daum, and P. Felke. On the security of HFE, HFEv- and Quartz. In Y. Desmedt, editor, *Proceedings of PKC 2003*, volume 2567 of *LNCS*, pages 337–50. Springer-Verlag, Jan. 2003.

[19] C. Diem. The GHS attack in odd characteristic. *J. Ramanujan Math. Soc.*, 18(1):1–32, 2003.

[20] R. Dupont, A. Enge, and F. Morain. Building curves with arbitrary small MOV degree over finite prime fields. *J. Cryptology*, 2005. To appear. Online: `http://www.springerlink.com/link.asp?id=mxqgw2mdwr5dtv34`.

[21] G. Frey, M. Muller, and H. Rück. The Tate pairing and the discrete logarithm applied to elliptic curve cryptosystems. *IEEE Trans. Info. Th.*, 45(5):1717–9, 1999.

[22] S. Galbraith. Supersingular curves in cryptography. In C. Boyd, editor, *Proceedings of Asiacrypt 2001*, volume 2248 of *LNCS*, pages 495–513. Springer-Verlag, Dec. 2001.

[23] S. Galbraith, K. Harrison, and D. Soldera. Implementing the Tate pairing. In C. Fieker and D. Kohel, editors, *Proceedings of ANTS V*, volume 2369 of *LNCS*, pages 324–37. Springer-Verlag, July 2002.

[24] S. Galbraith and N. Smart. A cryptographic application of Weil descent. In M. Walker, editor, *Proceedings of Cryptography and Coding 1999*, volume 1746 of *LNCS*, pages 191–200. Springer-Verlag, Dec. 1999.

[25] P. Gaudry, F. Hess, and N. Smart. Constructive and destructive facets of Weil descent on elliptic curves. *J. Cryptology*, 15(1):19–46, 2002.

[26] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In J. Stern, editor, *Proceedings of Eurocrypt 1999*, volume 1592 of *LNCS*, pages 295–310. Springer-Verlag, May 1999.

[27] E.-J. Goh and S. Jarecki. A signature scheme as secure as the Diffie-Hellman problem. In E. Biham, editor, *Proceedings of Eurocrypt 2003*, volume 2656 of *LNCS*, pages 401–15. Springer-Verlag, May 2003.

[28] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2):281–308, 1988.

[29] ISO TC86 Committee. ISO 8583: Financial transaction card originated messages — interchange message specifications. `http://www.tc68.org/`.

[30] A. Joux. A one round protocol for tripartite Diffie-Hellman. *J. Cryptology*, 17(4):263–76, Sept. 2004.

[31] A. Joux and K. Nguyen. Separating decision Diffie-Hellman from computational Diffie-Hellman in cryptographic groups. *J. Cryptology*, 16(4):239–47, Sept. 2003.

[32] J. Katz and N. Wang. Efficiency improvements for signature schemes with tight security reductions. In V. Atluri and T. Jaeger, editors, *Proceedings of CCS 2003*, pages 155–64. ACM Press, Oct. 2003.

[33] N. Koblitz. An elliptic curve implementation of the finite field digital signature algorithm. In H. Krawczyk, editor, *Proceedings of Crypto 1998*, volume 1462 of *LNCS*, pages 327–33. Springer-Verlag, Aug. 1998.

[34] S. Lang. *Elliptic Functions*. Addison-Wesley, Reading, MA, 1973.

[35] U. Maurer. Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete logarithms. In Y. Desmedt, editor, *Proceedings of Crypto 1994*, volume 839 of *LNCS*, pages 271–81. Springer-Verlag, Aug. 1994.

[36] A. Menezes, T. Okamoto, and P. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Trans. Info. Th.*, 39(5):1639–46, 1993.

[37] A. J. Menezes, editor. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, 1993.

[38] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[39] V. Miller. The Weil pairing, and its efficient calculation. *J. Cryptology*, 17(4):235–61, Sept. 2004.

[40] I. Mironov. A short signature as secure as DSA. Unpublished manuscript, 2001.

[41] A. Miyaji, M. Nakabayashi, and S. Takano. New explicit conditions of elliptic curve traces for FR-reduction. *IEICE Trans. Fundamentals*, E84-A(5):1234–43, May 2001.

[42] D. Naccache and J. Stern. Signing on a postcard. In Y. Frankel, editor, *Proceedings of Financial Cryptography 2000*, volume 1962 of *LNCS*, pages 121–35. Springer-Verlag, Feb. 2000.

[43] K. Nyberg and R. Rueppel. Message recovery for signature schemes based on the discrete logarithm problem. *Designs, Codes and Cryptography*, 7(1–2):61–81, 1996.

[44] T. Okamoto and D. Pointcheval. The gap problems: A new class of problems for the security of cryptographic primitives. In K. Kim, editor, *Proceedings of PKC 2001*, volume 1992 of *LNCS*, pages 104–18. Springer-Verlag, Feb. 2001.

[45] J. Patarin, N. Courtois, and L. Goubin. QUARTZ, 128-bit long digital signatures. In D. Naccache, editor, *Proceedings of CT-RSA 2001*, volume 2020 of *LNCS*, pages 282–97. Springer-Verlag, Apr. 2001.

[46] T. Pedersen. A threshold cryptosystem without a trusted third party. In D. W. Davies, editor, *Proceedings of Eurocrypt 1991*, volume 547 of *LNCS*, pages 522–6. Springer-Verlag, Apr. 1991.

[47] L. Pintsov and S. Vanstone. Postal revenue collection in the digital age. In Y. Frankel, editor, *Proceedings of Financial Cryptography 2000*, volume 1962 of *LNCS*, pages 105–20. Springer-Verlag, Feb. 2000.

[48] K. Rubin and A. Silverberg. Supersingular Abelian varieties in cryptology. In M. Yung, editor, *Proceedings of Crypto 2002*, volume 2442 of *LNCS*, pages 336–53. Springer-Verlag, Aug. 2002.

[49] O. Schirokauer, D. Weber, and T. Denny. Discrete logarithms: The effectiveness of the index calculus method. In H. Cohen, editor, *Proceedings of ANTS II*, volume 1122 of *LNCS*, pages 337–61. Springer-Verlag, May 1996.

[50] J. H. Silverman. *The Arithmetic of Elliptic Curves*, volume 106 of *Graduate Texts in Mathematics*. Springer-Verlag, 1986.

[51] N. Smart, editor. *The Algorithmic Resolution of Diophantine Equations*. Cambridge University Press, 1998.

[52] E. R. Verheul. Self-blindable credential certificates from the Weil pairing. In C. Boyd, editor, *Proceedings of Asiacrypt 2001*, volume 2248 of *LNCS*, pages 533–51. Springer-Verlag, Dec. 2001.

[53] E. R. Verheul. Evidence that XTR is more secure than supersingular elliptic curve cryptosystems. *J. Cryptology*, 17(4):277–96, Sept. 2004.

[54] W. C. Waterhouse. Abelian varieties over finite fields. *Ann. Sci. École Norm. Sup.*, 2:521–60, 1969.

Identity-Based Encryption from the Weil Pairing

Dan Boneh
dabo@cs.stanford.edu

Matthew Franklin
franklin@cs.ucdavis.edu

Abstract

We propose a fully functional identity-based encryption scheme (IBE). The scheme has chosen ciphertext security in the random oracle model assuming a variant of the computational Diffie-Hellman problem. Our system is based on bilinear maps between groups. The Weil pairing on elliptic curves is an example of such a map. We give precise definitions for secure identity based encryption schemes and give several applications for such systems.

1 Introduction

In 1984 Shamir [41] asked for a public key encryption scheme in which the public key can be an arbitrary string. In such a scheme there are four algorithms: (1) setup generates global system parameters and a master-key, (2) extract uses the master-key to generate the private key corresponding to an arbitrary public key string $\mathsf{ID} \in \{0,1\}^*$, (3) encrypt encrypts messages using the public key ID, and (4) decrypt decrypts messages using the corresponding private key.

Shamir's original motivation for identity-based encryption was to simplify certificate management in e-mail systems. When Alice sends mail to Bob at `bob@company.com` she simply encrypts her message using the public key string "`bob@company.com`". There is no need for Alice to obtain Bob's public key certificate. When Bob receives the encrypted mail he contacts a third party, which we call the Private Key Generator (PKG). Bob authenticates himself to the PKG in the same way he would authenticate himself to a CA and obtains his private key from the PKG. Bob can then read his e-mail. Note that unlike the existing secure e-mail infrastructure, Alice can send encrypted mail to Bob even if Bob has not yet setup his public key certificate. Also note that key escrow is inherent in identity-based e-mail systems: the PKG knows Bob's private key. We discuss key revocation, as well as several new applications for IBE schemes in the next section.

Since the problem was posed in 1984 there have been several proposals for IBE schemes [11, 45, 44, 31, 25] (see also [33, p. 561]). However, none of these are fully satisfactory. Some solutions require that users not collude. Other solutions require the PKG to spend a long time for each private key generation request. Some solutions require tamper resistant hardware. It is fair to say that until the results in [5] constructing a usable IBE system was an open problem. Interestingly, the related notions of identity-based signature and authentication schemes, also introduced by Shamir [41], do have satisfactory solutions [15, 14].

In this paper we propose a fully functional identity-based encryption scheme. The performance of our system is comparable to the performance of ElGamal encryption in $\mathbb{F}_p^*$. The security of our system is based on a natural analogue of the computational Diffie-Hellman assumption. Based on

this assumption we show that the new system has chosen ciphertext security in the random oracle model. Using standard techniques from threshold cryptography [20, 22] the PKG in our scheme can be distributed so that the master-key is never available in a single location. Unlike common threshold systems, we show that robustness for our distributed PKG is free.

Our IBE system can be built from any bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ between two groups $\mathbb{G}_1, \mathbb{G}_2$ as long as a variant of the Computational Diffie-Hellman problem in $\mathbb{G}_1$ is hard. We use the Weil pairing on elliptic curves as an example of such a map. Until recently the Weil pairing has mostly been used for attacking elliptic curve systems [32, 17]. Joux [26] recently showed that the Weil pairing can be used for "good" by using it for a protocol for three party one round Diffie-Hellman key exchange. Sakai et al. [40] used the pairing for key exchange and Verheul [46] used it to construct an ElGamal encryption scheme where each public key has two corresponding private keys. In addition to our identity-based encryption scheme, we show how to construct an ElGamal encryption scheme with "built-in" key escrow, i.e., where one global escrow key can decrypt ciphertexts encrypted under any public key.

To argue about the security of our IBE system we define chosen ciphertext security for identity-based encryption. Our model gives the adversary more power than the standard model for chosen ciphertext security [37, 2]. First, we allow the attacker to attack an arbitrary public key ID of her choice. Second, while mounting a chosen ciphertext attack on ID we allow the attacker to obtain from the PKG the private key for any public key of her choice, other than the private key for ID. This models an attacker who obtains a number of private keys corresponding to some identities of her choice and then tries to attack some other public key ID of her choice. Even with the help of such queries the attacker should have negligible advantage in defeating the semantic security of the system.

The rest of the paper is organized as follows. Several applications of identity-based encryption are discussed in Section 1.1. We then give precise definitions and security models in Section 2. We describe bilinear maps with certain properties in Section 3. Our identity-based encryption scheme is presented in Section 4 using general bilinear maps. Then a concrete identity based system from the Weil pairing is given in Section 5. Some extensions and variations (efficiency improvements, distribution of the master-key) are considered in Section 6. Our construction for ElGamal encryption with a global escrow key is described in Section 7. Section 8 gives conclusions and some open problems. The Appendix contains a more detailed discussion of the Weil pairing.

1.1 Applications for Identity-Based Encryption

The original motivation for identity-based encryption is to help the deployment of a public key infrastructure. In this section, we show several other unrelated applications.

1.1.1 Revocation of Public Keys

Public key certificates contain a preset expiration date. In an IBE system key expiration can be done by having Alice encrypt e-mail sent to Bob using the public key: "`bob@company.com` $\|$ `current-year`". In doing so Bob can use his private key during the current year only. Once a year Bob needs to obtain a new private key from the PKG. Hence, we get the effect of annual private key expiration. Note that unlike the existing PKI, Alice does not need to obtain a new certificate from Bob every time Bob refreshes his private key.

One could potentially make this approach more granular by encrypting e-mail for Bob using "`bob@company.com` $\|$ `current-date`". This forces Bob to obtain a new private key every day.

This might be possible in a corporate PKI where the PKG is maintained by the corporation. With this approach key revocation is very simple: when Bob leaves the company and his key needs to be revoked, the corporate PKG is instructed to stop issuing private keys for Bob's e-mail address. As a result, Bob can no longer read his email. The interesting property is that Alice does not need to communicate with any third party certificate directory to obtain Bob's daily public key. Hence, identity based encryption is a very efficient mechanism for implementing ephemeral public keys. Also note that this approach enables Alice to send messages into the future: Bob will only be able to decrypt the e-mail on the date specified by Alice (see [38, 12] for methods of sending messages into the future using a stronger security model).

Managing user credentials. A simple extension to the discussion above enables us to manage user credentials using the IBE system. Suppose Alice encrypts mail to Bob using the public key: "`bob@company.com` ‖ `current-year` ‖ `clearance=secret`". Then Bob will only be able to read the email if on the specified date he has secret clearance. Consequently, it is easy to grant and revoke user credentials using the PKG.

1.1.2 Delegation of Decryption Keys

Another application for IBE systems is delegation of decryption capabilities. We give two example applications. In both applications the user Bob plays the role of the PKG. Bob runs the setup algorithm to generate his own IBE system parameters params and his own master-key. Here we view params as Bob's public key. Bob obtains a certificate from a CA for his public key params. When Alice wishes to send mail to Bob she first obtains Bob's public key params from Bob's public key certificate. Note that Bob is the only one who knows his master-key and hence there is no key-escrow with this setup.

1. **Delegation to a laptop.** Suppose Alice encrypts mail to Bob using the current date as the IBE encryption key (she uses Bob's params as the IBE system parameters). Since Bob has the master-key he can extract the private key corresponding to this IBE encryption key and then decrypt the message. Now, suppose Bob goes on a trip for seven days. Normally, Bob would put his private key on his laptop. If the laptop is stolen the private key is compromised. When using the IBE system Bob could simply install on his laptop the seven private keys corresponding to the seven days of the trip. If the laptop is stolen, only the private keys for those seven days are compromised. The master-key is unharmed. This is analogous to the delegation scenario for *signature schemes* considered by Goldreich et al. [23].
2. **Delegation of duties.** Suppose Alice encrypts mail to Bob using the subject line as the IBE encryption key. Bob can decrypt mail using his master-key. Now, suppose Bob has several assistants each responsible for a different task (e.g. one is 'purchasing', another is 'human-resources', etc.). Bob gives one private key to each of his assistants corresponding to the assistant's responsibility. Each assistant can then decrypt messages whose subject line falls within its responsibilities, but it cannot decrypt messages intended for other assistants. Note that Alice only obtains a single public key from Bob (params), and she uses that public key to send mail with any subject line of her choice. The mail can only be read by the assistant responsible for that subject.

More generally, IBE can simplify security systems that manage a large number of public keys. Rather than storing a big database of public keys the system can either derive these public keys from usernames, or simply use the integers $1, \ldots, n$ as distinct public keys.

2 Definitions

Identity-Based Encryption. An identity-based encryption scheme $\mathcal{E}$ is specified by four randomized algorithms: Setup, Extract, Encrypt, Decrypt:

Setup: takes a security parameter k and returns params (system parameters) and master-key. The system parameters include a description of a finite message space $\mathcal{M}$, and a description of a finite ciphertext space $\mathcal{C}$. Intuitively, the system parameters will be publicly known, while the master-key will be known only to the "Private Key Generator" (PKG).

Extract: takes as input params, master-key, and an arbitrary $\mathsf{ID} \in \{0,1\}^*$, and returns a private key d. Here ID is an arbitrary string that will be used as a public key, and d is the corresponding private decryption key. The Extract algorithm extracts a private key from the given public key.

Encrypt: takes as input params, ID, and $M \in \mathcal{M}$. It returns a ciphertext $C \in \mathcal{C}$.

Decrypt: takes as input params, $C \in \mathcal{C}$, and a private key d. It return $M \in \mathcal{M}$.

These algorithms must satisfy the standard consistency constraint, namely when d is the private key generated by algorithm Extract when it is given ID as the public key, then

$$\forall M \in \mathcal{M} : \mathsf{Decrypt}(\mathsf{params}, C, d) = M \quad \text{where} \quad C = \mathsf{Encrypt}(\mathsf{params}, \mathsf{ID}, M)$$

Chosen ciphertext security. Chosen ciphertext security (IND-CCA) is the standard acceptable notion of security for a public key encryption scheme [37, 2, 13]. Hence, it is natural to require that an identity-based encryption scheme also satisfy this strong notion of security. However, the definition of chosen ciphertext security must be strengthened a bit. The reason is that when an adversary attacks a public key ID in an identity-based system, the adversary might already possess the private keys of users $\mathsf{ID}_1, \ldots, \mathsf{ID}_n$ of her choice. The system should remain secure under such an attack. Hence, the definition of chosen ciphertext security must allow the adversary to obtain the private key associated with any identity ID_i of her choice (other than the public key ID being attacked). We refer to such queries as private key extraction queries. Another difference is that the adversary is challenged on a public key ID of her choice (as opposed to a random public key).

We say that an identity-based encryption scheme $\mathcal{E}$ is semantically secure against an adaptive chosen ciphertext attack (IND-ID-CCA) if no polynomially bounded adversary $\mathcal{A}$ has a non-negligible advantage against the Challenger in the following IND-ID-CCA game:

Setup: The challenger takes a security parameter k and runs the Setup algorithm. It gives the adversary the resulting system parameters params. It keeps the master-key to itself.

Phase 1: The adversary issues queries $q_1, \ldots, q_m$ where query q_i is one of:

- Extraction query $\langle \mathsf{ID}_i \rangle$. The challenger responds by running algorithm Extract to generate the private key d_i corresponding to the public key $\langle \mathsf{ID}_i \rangle$. It sends d_i to the adversary.
- Decryption query $\langle \mathsf{ID}_i, C_i \rangle$. The challenger responds by running algorithm Extract to generate the private key d_i corresponding to ID_i. It then runs algorithm Decrypt to decrypt the ciphertext C_i using the private key d_i. It sends the resulting plaintext to the adversary.

These queries may be asked adaptively, that is, each query q_i may depend on the replies to $q_1, \ldots, q_{i-1}$.

Challenge: Once the adversary decides that Phase 1 is over it outputs two equal length plaintexts $M_0, M_1 \in \mathcal{M}$ and an identity ID on which it wishes to be challenged. The only constraint is that ID did not appear in any private key extraction query in Phase 1.

The challenger picks a random bit $b \in \{0,1\}$ and sets $C = \mathsf{Encrypt}(\mathsf{params}, \mathsf{ID}, M_b)$. It sends C as the challenge to the adversary.

Phase 2: The adversary issues more queries $q_{m+1}, \ldots, q_n$ where query q_i is one of:

– Extraction query $\langle \mathsf{ID}_i \rangle$ where $\mathsf{ID}_i \neq \mathsf{ID}$. Challenger responds as in Phase 1.

– Decryption query $\langle \mathsf{ID}_i, C_i \rangle \neq \langle \mathsf{ID}, C \rangle$. Challenger responds as in Phase 1.

These queries may be asked adaptively as in Phase 1.

Guess: Finally, the adversary outputs a guess $b' \in \{0,1\}$ and wins the game if $b = b'$.

We refer to such an adversary $\mathcal{A}$ as an IND-ID-CCA adversary. We define adversary $\mathcal{A}$'s advantage in attacking the scheme $\mathcal{E}$ as the following function of the security parameter k (k is given as input to the challenger): $\mathrm{Adv}_{\mathcal{E},\mathcal{A}}(k) = \left|\Pr[b = b'] - \frac{1}{2}\right|$.
The probability is over the random bits used by the challenger and the adversary.

Using the IND-ID-CCA game we can define chosen ciphertext security for IBE schemes. As usual, we say that a function $g : \mathbb{R} \to \mathbb{R}$ is *negligible* if for any $d > 0$ we have $|g(k)| < 1/k^d$ for sufficiently large k.

Definition 2.1. *We say that the IBE system $\mathcal{E}$ is semantically secure against an adaptive chosen ciphertext attack if for any polynomial time* IND-ID-CCA *adversary $\mathcal{A}$ the function $Adv_{\mathcal{E},\mathcal{A}}(k)$ is negligible. As shorthand, we say that $\mathcal{E}$ is* IND-ID-CCA *secure.*

Note that the standard definition of chosen ciphertext security (IND-CCA) [37, 2] is the same as above except that there are no private key extraction queries and the adversary is challenged on a random public key (rather than a public key of her choice). Private key extraction queries are related to the definition of chosen ciphertext security in the multiuser settings [7]. After all, our definition involves multiple public keys belonging to multiple users. In [7] the authors show that that multiuser IND-CCA is reducible to single user IND-CCA using a standard hybrid argument. This does not hold in the identity-based settings, IND-ID-CCA, since the adversary gets to choose which public keys to corrupt during the attack. To emphasize the importance of private key extraction queries we note that our IBE system can be easily modified (by removing one of the hash functions) into a system which has chosen ciphertext security when private extraction queries are disallowed. However, the scheme is completely insecure when extraction queries are allowed.

Semantically secure identity based encryption. The proof of security for our IBE system makes use of a weaker notion of security known as semantic security (also known as semantic security against a chosen plaintext attack) [24, 2]. Semantic security is similar to chosen ciphertext security (IND-ID-CCA) except that the adversary is more limited; it cannot issue decryption queries while attacking the challenge public key. For a standard public key system (not an identity based system) semantic security is defined using the following game: (1) the adversary is given a random public key generated by the challenger, (2) the adversary outputs two equal length messages M_0 and M_1 and receives the encryption of M_b from the challenger where b is chosen at random in $\{0,1\}$, (3) the adversary outputs b' and wins the game if $b = b'$. The public key system is said to be semantically secure if no polynomial time adversary can win the game with a non-negligible advantage. As shorthand we say that a semantically secure public key system is IND-CPA secure. Semantic security captures our intuition that given a ciphertext the adversary learns nothing about the corresponding plaintext.

To define semantic security for identity based systems (denoted IND-ID-CPA) we strengthen the standard definition by allowing the adversary to issue chosen private key extraction queries. Similarly, the adversary is challenged on a public key ID of her choice. We define semantic security for identity based encryption schemes using an IND-ID-CPA game. The game is identical to the IND-ID-CCA game defined above except that the adversary cannot make any decryption queries. The adversary can only make private key extraction queries. We say that an identity-based encryption scheme $\mathcal{E}$ is semantically secure (IND-ID-CPA) if no polynomially bounded adversary $\mathcal{A}$ has a non-negligible advantage against the Challenger in the following IND-ID-CPA game:

Setup: The challenger takes a security parameter k and runs the Setup algorithm. It gives the adversary the resulting system parameters params. It keeps the master-key to itself.

Phase 1: The adversary issues private key extraction queries $\mathsf{ID}_1, \ldots, \mathsf{ID}_m$. The challenger responds by running algorithm Extract to generate the private key d_i corresponding to the public key ID_i. It sends d_i to the adversary. These queries may be asked adaptively.

Challenge: Once the adversary decides that Phase 1 is over it outputs two equal length plaintexts $M_0, M_1 \in \mathcal{M}$ and a public key ID on which it wishes to be challenged. The only constraint is that ID did not appear in any private key extraction query in Phase 1. The challenger picks a random bit $b \in \{0,1\}$ and sets $C = \mathsf{Encrypt}(\mathsf{params}, \mathsf{ID}, M_b)$. It sends C as the challenge to the adversary.

Phase 2: The adversary issues more extraction queries $\mathsf{ID}_{m+1}, \ldots, \mathsf{ID}_n$. The only constraint is that $\mathsf{ID}_i \neq \mathsf{ID}$. The challenger responds as in Phase 1.

Guess: Finally, the adversary outputs a guess $b' \in \{0,1\}$ and wins the game if $b = b'$.

We refer to such an adversary $\mathcal{A}$ as an IND-ID-CPA adversary. As we did above, the advantage of an IND-ID-CPA adversary $\mathcal{A}$ against the scheme $\mathcal{E}$ is the following function of the security parameter k: $\mathrm{Adv}_{\mathcal{E},\mathcal{A}}(k) = \left|\Pr[b = b'] - \frac{1}{2}\right|$.
The probability is over the random bits used by the challenger and the adversary.

Definition 2.2. *We say that the IBE system $\mathcal{E}$ is semantically secure if for any polynomial time* IND-ID-CPA *adversary $\mathcal{A}$ the function $Adv_{\mathcal{E},\mathcal{A}}(k)$ is negligible. As shorthand, we say that $\mathcal{E}$ is* IND-ID-CPA *secure.*

One way identity-based encryption. One can define an even weaker notion of security called one-way encryption (OWE) [16]. Roughly speaking, a public key encryption scheme is a one-way encryption if given the encryption of a random plaintext the adversary cannot produce the plaintext in its entirety. One way encryption is a weak notion of security since there is nothing preventing the adversary from, say, learning half the bits of the plaintext. Hence, one-way encryption schemes do not generally provide secure encryption. In the random oracle model one-way encryption schemes can be used for encrypting session-keys (the session-key is taken to be the hash of the plaintext). We note that one can extend the notion of one-way encryption to identity based systems by adding private key extraction queries to the definition. We do not give the full definition here since in this paper we use semantic security as the weakest notion of security. See [5] for the full definition of identity based one-way encryption, and its use as part of an alternative proof strategy for our main result.

Random oracle model. To analyze the security of certain natural cryptographic constructions Bellare and Rogaway introduced an idealized security model called the random oracle model [3]. Roughly

speaking, a random oracle is a function $H : X \to Y$ chosen uniformly at random from the set of all functions $\{h : X \to Y\}$ (we assume Y is a finite set). An algorithm can query the random oracle at any point $x \in X$ and receive the value $H(x)$ in response. Random oracles are used to model cryptographic hash functions such as SHA-1. Note that security in the random oracle model does not imply security in the real world. Nevertheless, the random oracle model is a useful tool for validating natural cryptographic constructions. Security proofs in this model prove security against attackers that are confined to the random oracle world.

Notation. From here on we use $\mathbb{Z}_q$ to denote the group $\{0, \ldots, q-1\}$ under addition modulo q. For a group $\mathbb{G}$ of prime order we use $\mathbb{G}^*$ to denote the set $\mathbb{G}^* = \mathbb{G} \setminus \{O\}$ where O is the identity element in the group $\mathbb{G}$. We use $\mathbb{Z}^+$ to denote the set of positive integers.

3 Bilinear maps and the Bilinear Diffie-Hellman Assumption

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two groups of order q for some large prime q. Our IBE system makes use of a *bilinear* map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ between these two groups. The map must satisfy the following properties:

1. Bilinear: We say that a map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ is *bilinear* if $\hat{e}(aP, bQ) = \hat{e}(P, Q)^{ab}$ for all $P, Q \in \mathbb{G}_1$ and all $a, b \in \mathbb{Z}$.
2. Non-degenerate: The map does not send all pairs in $\mathbb{G}_1 \times \mathbb{G}_1$ to the identity in $\mathbb{G}_2$. Observe that since $\mathbb{G}_1, \mathbb{G}_2$ are groups of prime order this implies that if P is a generator of $\mathbb{G}_1$ then $\hat{e}(P, P)$ is a generator of $\mathbb{G}_2$.
3. Computable: There is an efficient algorithm to compute $\hat{e}(P, Q)$ for any $P, Q \in \mathbb{G}_1$.

A bilinear map satisfying the three properties above is said to be an *admissible* bilinear map. In Section 5 we give a concrete example of groups $\mathbb{G}_1, \mathbb{G}_2$ and an admissible bilinear map between them. The group $\mathbb{G}_1$ is a subgroup of the additive group of points of an elliptic curve $E/\mathbb{F}_p$. The group $\mathbb{G}_2$ is a subgroup of the multiplicative group of a finite field $\mathbb{F}^*_{p^2}$. Therefore, throughout the paper we view $\mathbb{G}_1$ as an additive group and $\mathbb{G}_2$ as a multiplicative group. As we will see in Section 5.1, the Weil pairing can be used to construct an admissible bilinear map between these two groups.

The existence of the bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ as above has two direct implications to these groups.

The MOV reduction: Menezes, Okamoto, and Vanstone [32] show that the discrete log problem in $\mathbb{G}_1$ is no harder than the discrete log problem in $\mathbb{G}_2$. To see this, let $P, Q \in \mathbb{G}_1$ be an instance of the discrete log problem in $\mathbb{G}_1$ where both P, Q have order q. We wish to find an $\alpha \in \mathbb{Z}_q$ such that $Q = \alpha P$. Let $g = \hat{e}(P, P)$ and $h = \hat{e}(Q, P)$. Then, by bilinearity of $\hat{e}$ we know that $h = g^\alpha$. By non-degeneracy of $\hat{e}$ both g, h have order q in $\mathbb{G}_2$. Hence, we reduced the discrete log problem in $\mathbb{G}_1$ to a discrete log problem in $\mathbb{G}_2$. It follows that for discrete log to be hard in $\mathbb{G}_1$ we must choose our security parameter so that discrete log is hard in $\mathbb{G}_2$ (see Section 5).

Decision Diffie-Hellman is Easy: The Decision Diffie-Hellman problem (DDH) [4] in $\mathbb{G}_1$ is to distinguish between the distributions $\langle P, aP, bP, abP\rangle$ and $\langle P, aP, bP, cP\rangle$ where a, b, c are random in $\mathbb{Z}^*_q$ and P is random in $\mathbb{G}^*_1$. Joux and Nguyen [28] point out that DDH in $\mathbb{G}_1$ is easy. To see this, observe that given $P, aP, bP, cP \in \mathbb{G}^*_1$ we have

$$c = ab \bmod q \quad \Longleftrightarrow \quad \hat{e}(P, cP) = \hat{e}(aP, bP).$$

The Computational Diffie-Hellman problem (CDH) in $\mathbb{G}_1$ can still be hard (CDH in G_1 is to find abP given random $\langle P, aP, bP\rangle$). Joux and Nguyen [28] give examples of mappings $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ where CDH in $\mathbb{G}_1$ is believed to be hard even though DDH in $\mathbb{G}_1$ is easy.

3.1 The Bilinear Diffie-Hellman Assumption (BDH)

Since the Decision Diffie-Hellman problem (DDH) in $\mathbb{G}_1$ is easy we cannot use DDH to build cryptosystems in the group $\mathbb{G}_1$. Instead, the security of our IBE system is based on a variant of the Computational Diffie-Hellman assumption called the Bilinear Diffie-Hellman Assumption (BDH).

Bilinear Diffie-Hellman Problem. Let $\mathbb{G}_1, \mathbb{G}_2$ be two groups of prime order q. Let $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ be an admissible bilinear map and let P be a generator of $\mathbb{G}_1$. The BDH problem in $\langle \mathbb{G}_1, \mathbb{G}_2, \hat{e}\rangle$ is as follows: Given $\langle P, aP, bP, cP\rangle$ for some $a, b, c \in \mathbb{Z}_q^*$ compute $W = \hat{e}(P, P)^{abc} \in \mathbb{G}_2$. An algorithm $\mathcal{A}$ has advantage ϵ in solving BDH in $\langle \mathbb{G}_1, \mathbb{G}_2, \hat{e}\rangle$ if

$$\Pr\left[\mathcal{A}(P, aP, bP, cP) = \hat{e}(P, P)^{abc}\right] \geq \epsilon$$

where the probability is over the random choice of a, b, c in $\mathbb{Z}_q^*$, the random choice of $P \in \mathbb{G}_1^*$, and the random bits of $\mathcal{A}$.

BDH Parameter Generator. We say that a randomized algorithm $\mathcal{G}$ is a *BDH parameter generator* if (1) $\mathcal{G}$ takes a security parameter $k \in \mathbb{Z}^+$, (2) $\mathcal{G}$ runs in polynomial time in k, and (3) $\mathcal{G}$ outputs a prime number q, the description of two groups $\mathbb{G}_1, \mathbb{G}_2$ of order q, and the description of an admissible bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$. We denote the output of $\mathcal{G}$ by $\mathcal{G}(1^k) = \langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}\rangle$. The security parameter k is used to determine the size of q; for example, one could take q to be a random k-bit prime. For $i = 1, 2$ we assume that the description of the group $\mathbb{G}_i$ contains polynomial time (in k) algorithms for computing the group action in $\mathbb{G}_i$ and contains a generator of $\mathbb{G}_i$. The generator of $\mathbb{G}_i$ enables us to generate uniformly random elements in $\mathbb{G}_i$. Similarly, we assume that the description of $\hat{e}$ contains a polynomial time algorithm for computing $\hat{e}$. We give an example of a BDH parameter generator in Section 5.1.

BDH Assumption. Let $\mathcal{G}$ be a BDH parameter generator. We say that an algorithm $\mathcal{A}$ has advantage $\epsilon(k)$ in solving the BDH problem for $\mathcal{G}$ if for sufficiently large k:

$$\mathrm{Adv}_{\mathcal{G},\mathcal{A}}(k) = \Pr\left[\mathcal{A}(q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, aP, bP, cP) = \hat{e}(P, P)^{abc} \;\middle|\; \begin{array}{l} \langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}\rangle \leftarrow \mathcal{G}(1^k), \\ P \leftarrow \mathbb{G}_1^*,\ a, b, c \leftarrow \mathbb{Z}_q^* \end{array}\right] \geq \epsilon(k)$$

We say that $\mathcal{G}$ satisfies the BDH assumption if for any randomized polynomial time (in k) algorithm $\mathcal{A}$ we have that $\mathrm{Adv}_{\mathcal{G},\mathcal{A}}(k)$ is a negligible function. When $\mathcal{G}$ satisfies the BDH assumption we say that BDH is hard in groups generated by $\mathcal{G}$.

In Section 5.1 we give some examples of BDH parameter generators that are believed to satisfy the BDH assumption. We note that Joux [26] (implicitly) used the BDH assumption to construct a one-round three party Diffie-Hellman protocol. The BDH assumption is also needed for constructions in [46, 40].

Hardness of BDH. It is interesting to study the relationship of the BDH problem to other hard problems used in cryptography. Currently, all we can say is that the BDH problem in $\langle \mathbb{G}_1, \mathbb{G}_2, \hat{e} \rangle$ is no harder than the CDH problem in $\mathbb{G}_1$ or $\mathbb{G}_2$. In other words, an algorithm for CDH in $\mathbb{G}_1$ or $\mathbb{G}_2$ is sufficient for solving BDH in $\langle \mathbb{G}_1, \mathbb{G}_2, \hat{e} \rangle$. The converse is currently an open problem: is an algorithm for BDH sufficient for solving CDH in $\mathbb{G}_1$ or in $\mathbb{G}_2$? We refer to a survey by Joux [27] for a more detailed analysis of the relationship between BDH and other standard problems.

We note that in all our examples (in Section 5.1) the isomorphisms from $\mathbb{G}_1$ to $\mathbb{G}_2$ induced by the bilinear map are believed to be one-way functions. More specifically, for a point $Q \in \mathbb{G}_1^*$ define the isomorphism $f_Q : \mathbb{G}_1 \to \mathbb{G}_2$ by $f_Q(P) = \hat{e}(P, Q)$. If any one of these isomorphisms turns out to be invertible then BDH is easy in $\langle \mathbb{G}_1, \mathbb{G}_2, \hat{e} \rangle$. Fortunately, an efficient algorithm for inverting f_Q for some fixed Q would imply an efficient algorithm for deciding DDH in the group $\mathbb{G}_2$. In all our examples DDH is believed to be hard in the group $\mathbb{G}_2$. Hence, all the isomorphisms $f_Q : \mathbb{G}_1 \to \mathbb{G}_2$ induced by the bilinear map are believed to be one-way functions.

4 Our Identity-Based Encryption Scheme

We describe our scheme in stages. First we give a basic identity-based encryption scheme which is not secure against an adaptive chosen ciphertext attack. The only reason for describing the basic scheme is to make the presentation easier to follow. Our full scheme, described in Section 4.2, extends the basic scheme to get security against an adaptive chosen ciphertext attack (IND-ID-CCA) in the random oracle model. In Section 4.3 we relax some of the requirements on the hash functions.

The presentation in this section uses an arbitrary BDH parameter generator $\mathcal{G}$ satisfying the BDH assumption. In Section 5 we describe a concrete IBE system using the Weil pairing.

4.1 BasicIdent

To explain the basic ideas underlying our IBE system we describe the following simple scheme, called BasicIdent. We present the scheme by describing the four algorithms: Setup, Extract, Encrypt, Decrypt. We let k be the security parameter given to the setup algorithm. We let $\mathcal{G}$ be some BDH parameter generator.

Setup: Given a security parameter $k \in \mathbb{Z}^+$, the algorithm works as follows:

Step 1: Run $\mathcal{G}$ on input k to generate a prime q, two groups $\mathbb{G}_1, \mathbb{G}_2$ of order q, and an admissible bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$. Choose a random generator $P \in \mathbb{G}_1$.

Step 2: Pick a random $s \in \mathbb{Z}_q^*$ and set $P_{pub} = sP$.

Step 3: Choose a cryptographic hash function $H_1 : \{0,1\}^* \to \mathbb{G}_1^*$. Choose a cryptographic hash function $H_2 : \mathbb{G}_2 \to \{0,1\}^n$ for some n. The security analysis will view H_1, H_2 as random oracles.

The message space is $\mathcal{M} = \{0,1\}^n$. The ciphertext space is $\mathcal{C} = \mathbb{G}_1^* \times \{0,1\}^n$. The system parameters are $\mathsf{params} = \langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, n, P, P_{pub}, H_1, H_2 \rangle$. The master-key is $s \in \mathbb{Z}_q^*$.

Extract: For a given string $\mathsf{ID} \in \{0,1\}^*$ the algorithm does: (1) computes $Q_{\mathsf{ID}} = H_1(\mathsf{ID}) \in \mathbb{G}_1^*$, and (2) sets the private key d_{ID} to be $d_{\mathsf{ID}} = sQ_{\mathsf{ID}}$ where s is the master key.

Encrypt: To encrypt $M \in \mathcal{M}$ under the public key ID do the following: (1) compute $Q_{\mathsf{ID}} = H_1(\mathsf{ID}) \in \mathbb{G}_1^*$, (2) choose a random $r \in \mathbb{Z}_q^*$, and (3) set the ciphertext to be

$$C = \langle rP,\ \ M \oplus H_2(g_{\mathsf{ID}}^r) \rangle \quad \text{where} \quad g_{\mathsf{ID}} = \hat{e}(Q_{\mathsf{ID}}, P_{pub}) \in \mathbb{G}_2^*$$

Decrypt: Let $C = \langle U, V \rangle \in \mathcal{C}$ be a ciphertext encrypted using the public key ID. To decrypt C using the private key $d_{\mathsf{ID}} \in \mathbb{G}_1^*$ compute:

$$V \oplus H_2(\hat{e}(d_{\mathsf{ID}}, U)) = M$$

This completes the description of BasicIdent. We first verify consistency. When everything is computed as above we have:
1. During encryption M is bitwise exclusive-ored with the hash of: g_{ID}^r.
2. During decryption V is bitwise exclusive-ored with the hash of: $\hat{e}(d_{\mathsf{ID}}, U)$.
These masks used during encryption and decryption are the same since:

$$\hat{e}(d_{\mathsf{ID}}, U) = \hat{e}(sQ_{\mathsf{ID}}, rP) = \hat{e}(Q_{\mathsf{ID}}, P)^{sr} = \hat{e}(Q_{\mathsf{ID}}, P_{pub})^r = g_{\mathsf{ID}}^r$$

Thus, applying decryption after encryption produces the original message M as required. Performance considerations of BasicIdent are discussed in Section 5. Note that the value of g_{ID} in Algorithm Encrypt is independent of the message to be encrypted. Hence there is no need to recompute g_{ID} on subsequent encryptions to the same public key ID.

Security. Next, we study the security of this basic scheme. The following theorem shows that BasicIdent is a semantically secure identity based encryption scheme (IND-ID-CPA) assuming BDH is hard in groups generated by $\mathcal{G}$.

Theorem 4.1. *Suppose the hash functions H_1, H_2 are random oracles. Then* BasicIdent *is a semantically secure identity based encryption scheme (*IND-ID-CPA*) assuming BDH is hard in groups generated by $\mathcal{G}$. Concretely, suppose there is an* IND-ID-CPA *adversary $\mathcal{A}$ that has advantage $\epsilon(k)$ against the scheme* BasicIdent*. Suppose $\mathcal{A}$ makes at most $q_E > 0$ private key extraction queries and $q_{H_2} > 0$ hash queries to H_2. Then there is an algorithm $\mathcal{B}$ that solves BDH in groups generated by $\mathcal{G}$ with advantage at least:*

$$Adv_{\mathcal{G},\mathcal{B}}(k) \geq \frac{2\epsilon(k)}{e(1 + q_E) \cdot q_{H_2}}$$

Here $e \approx 2.71$ is the base of the natural logarithm. The running time of $\mathcal{B}$ is $O(time(\mathcal{A}))$.

To prove the theorem we first define a related Public Key Encryption scheme (not an identity based scheme), called BasicPub. BasicPub is described by three algorithms: keygen, encrypt, decrypt.

keygen: Given a security parameter $k \in \mathbb{Z}^+$, the algorithm works as follows:

Step 1: Run $\mathcal{G}$ on input k to generate two prime order groups $\mathbb{G}_1, \mathbb{G}_2$ and a bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$. Let q be the order of $\mathbb{G}_1, \mathbb{G}_2$. Choose a random generator $P \in \mathbb{G}_1$.

Step 2: Pick a random $s \in \mathbb{Z}_q^*$ and set $P_{pub} = sP$. Pick a random $Q_{\mathsf{ID}} \in \mathbb{G}_1^*$.

Step 3: Choose a cryptographic hash function $H_2 : \mathbb{G}_2 \rightarrow \{0,1\}^n$ for some n.

Step 4: The public key is $\langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, n, P, P_{pub}, Q_{\mathsf{ID}}, H_2 \rangle$. The private key is $d_{\mathsf{ID}} = sQ_{\mathsf{ID}} \in \mathbb{G}_1^*$.

encrypt: To encrypt $M \in \{0,1\}^n$ choose a random $r \in \mathbb{Z}_q^*$ and set the ciphertext to be:

$$C = \langle rP, \;\; M \oplus H_2(g^r) \rangle \quad \text{where} \quad g = \hat{e}(Q_{\mathsf{ID}}, P_{pub}) \in \mathbb{G}_2^*$$

decrypt: Let $C = \langle U, V \rangle$ be a ciphertext created using the public key $\langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, n, P, P_{pub}, Q_{\mathsf{ID}}, H_2 \rangle$. To decrypt C using the private key $d_{\mathsf{ID}} \in \mathbb{G}_1^*$ compute:

$$V \oplus H_2(\hat{e}(d_{\mathsf{ID}}, U)) = M$$

This completes the description of BasicPub. We now prove Theorem 4.1 in two steps. We first show that an IND-ID-CPA attack on BasicIdent can be converted to a IND-CPA attack on BasicPub. This step shows that private key extraction queries do not help the adversary. We then show that BasicPub is IND-CPA secure if the BDH assumption holds.

Lemma 4.2. *Let H_1 be a random oracle from $\{0,1\}^*$ to $\mathbb{G}_1^*$. Let $\mathcal{A}$ be an* IND-ID-CPA *adversary that has advantage $\epsilon(k)$ against* BasicIdent*. Suppose $\mathcal{A}$ makes at most $q_E > 0$ private key extraction queries. Then there is a* IND-CPA *adversary $\mathcal{B}$ that has advantage at least $\epsilon(k)/e(1+q_E)$ against* BasicPub*. Its running time is $O(time(\mathcal{A}))$.*

Proof. We show how to construct an IND-CPA adversary $\mathcal{B}$ that uses $\mathcal{A}$ to gain advantage $\epsilon/e(1+q_E)$ against BasicPub. The game between the challenger and the adversary $\mathcal{B}$ starts with the challenger first generating a random public key by running algorithm keygen of BasicPub. The result is a public key $K_{pub} = \langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, n, P, P_{pub}, Q_{\mathsf{ID}}, H_2\rangle$ and a private key $d_{\mathsf{ID}} = sQ_{\mathsf{ID}}$. As usual, q is the order of $\mathbb{G}_1, \mathbb{G}_2$. The challenger gives K_{pub} to algorithm $\mathcal{B}$. Algorithm $\mathcal{B}$ is supposed to output two messages M_0 and M_1 and expects to receive back the BasicPub encryption of M_b under K_{pub} where $b \in \{0,1\}$. Then algorithm $\mathcal{B}$ outputs its guess $b' \in \{0,1\}$ for b.

Algorithm $\mathcal{B}$ works by interacting with $\mathcal{A}$ in an IND-ID-CPA game as follows ($\mathcal{B}$ simulates the challenger for $\mathcal{A}$):

Setup: Algorithm $\mathcal{B}$ gives $\mathcal{A}$ the BasicIdent system parameters $\langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, n, P, P_{pub}, H_1, H_2\rangle$. Here $q, \mathbb{G}_1, \mathbb{G}_2$, $\hat{e}$, n, P, P_{pub}, H_2 are taken from K_{pub}, and H_1 is a random oracle controlled by $\mathcal{B}$ as described below.

H_1-queries: At any time algorithm $\mathcal{A}$ can query the random oracle H_1. To respond to these queries algorithm $\mathcal{B}$ maintains a list of tuples $\langle \mathsf{ID}_j, Q_j, b_j, c_j\rangle$ as explained below. We refer to this list as the H_1^{list}. The list is initially empty. When $\mathcal{A}$ queries the oracle H_1 at a point ID_i algorithm $\mathcal{B}$ responds as follows:

1. If the query ID_i already appears on the H_1^{list} in a tuple $\langle \mathsf{ID}_i, Q_i, b_i, c_i\rangle$ then Algorithm $\mathcal{B}$ responds with $H_1(\mathsf{ID}_i) = Q_i \in \mathbb{G}_1^*$.
2. Otherwise, $\mathcal{B}$ generates a random $coin \in \{0,1\}$ so that $\Pr[coin = 0] = \delta$ for some δ that will be determined later.
3. Algorithm $\mathcal{B}$ picks a random $b \in \mathbb{Z}_q^*$.
   If $coin = 0$ compute $Q_i = bP \in \mathbb{G}_1^*$. If $coin = 1$ compute $Q_i = bQ_{\mathsf{ID}} \in \mathbb{G}_1^*$.
4. Algorithm $\mathcal{B}$ adds the tuple $\langle \mathsf{ID}_i, Q_i, b, coin\rangle$ to the H_1^{list} and responds to $\mathcal{A}$ with $H_1(\mathsf{ID}_i) = Q_i$.

Note that either way Q_i is uniform in $\mathbb{G}_1^*$ and is independent of $\mathcal{A}$'s current view as required.

Phase 1: Let ID_i be a private key extraction query issued by algorithm $\mathcal{A}$. Algorithm $\mathcal{B}$ responds to this query as follows:

1. Run the above algorithm for responding to H_1-queries to obtain a $Q_i \in \mathbb{G}_1^*$ such that $H_1(\mathsf{ID}_i) = Q_i$. Let $\langle \mathsf{ID}_i, Q_i, b_i, coin_i\rangle$ be the corresponding tuple on the H_1^{list}. If $coin_i = 1$ then $\mathcal{B}$ reports failure and terminates. The attack on BasicPub failed.
2. We know $coin_i = 0$ and hence $Q_i = b_iP$. Define $d_i = b_iP_{pub} \in \mathbb{G}_1^*$. Observe that $d_i = sQ_i$ and therefore d_i is the private key associated to the public key ID_i. Give d_i to algorithm $\mathcal{A}$.

Challenge: Once algorithm $\mathcal{A}$ decides that Phase 1 is over it outputs a public key ID_{ch} and two messages M_0, M_1 on which it wishes to be challenged. Algorithm $\mathcal{B}$ responds as follows:

1. Algorithm $\mathcal{B}$ gives its challenger the messages M_0, M_1. The challenger responds with a BasicPub ciphertext $C = \langle U, V\rangle$ such that C is the encryption of M_c for a random $c \in \{0,1\}$.
2. Next, $\mathcal{B}$ runs the algorithm for responding to H_1-queries to obtain a $Q \in \mathbb{G}_1^*$ such that $H_1(\mathsf{ID}_{ch}) =$

Q. Let $\langle \mathsf{ID}_{ch}, Q, b, coin\rangle$ be the corresponding tuple on the H_1^{list}. If $coin = 0$ then $\mathcal{B}$ reports failure and terminates. The attack on BasicPub failed.

3. We know $coin = 1$ and therefore $Q = bQ_{\mathsf{ID}}$. Recall that when $C = \langle U, V\rangle$ we have $U \in \mathbb{G}_1^*$. Set $C' = \langle b^{-1}U,\ V\rangle$, where b^{-1} is the inverse of $b \bmod q$. Algorithm $\mathcal{B}$ responds to $\mathcal{A}$ with the challenge ciphertext C'. Note that C' is a proper BasicIdent encryption of M_c under the public key ID_{ch} as required. To see this first observe that, since $H_1(\mathsf{ID}_{ch}) = Q$, the private key corresponding to ID_{ch} is $d_{ch} = sQ$. Second, observe that

$$\hat{e}(b^{-1}U, d_{ch}) = \hat{e}(b^{-1}U, sQ) = \hat{e}(U, sb^{-1}Q) = \hat{e}(U, sQ_{\mathsf{ID}}) = \hat{e}(U, d_{\mathsf{ID}}).$$

Hence, the BasicIdent decryption of C' using d_{ch} is the same as the BasicPub decryption of C using d_{ID}.

Phase 2: Algorithm $\mathcal{B}$ responds to private key extraction queries as in Phase 1.

Guess: Eventually algorithm $\mathcal{A}$ outputs a guess c' for c. Algorithm $\mathcal{B}$ outputs c' as its guess for c.

Claim: If algorithm $\mathcal{B}$ does not abort during the simulation then algorithm $\mathcal{A}$'s view is identical to its view in the real attack. Furthermore, if $\mathcal{B}$ does not abort then $|\Pr[c = c'] - \frac{1}{2}| \geq \epsilon$. The probability is over the random bits used by $\mathcal{A}, \mathcal{B}$ and the challenger.

Proof of claim. The responses to H_1-queries are as in the real attack since each response is uniformly and independently distributed in $\mathbb{G}_1^*$. All responses to private key extraction queries are valid. Finally, the challenge ciphertext C' given to $\mathcal{A}$ is the BasicIdent encryption of M_c for some random $c \in \{0, 1\}$. Therefore, by definition of algorithm $\mathcal{A}$ we have that $|\Pr[c = c'] - \frac{1}{2}| \geq \epsilon$. □

To complete the proof of Lemma 4.2 it remains to calculate the probability that algorithm $\mathcal{B}$ aborts during the simulation. Suppose $\mathcal{A}$ makes a total of q_E private key extraction queries. Then the probability that $\mathcal{B}$ does not abort in phases 1 or 2 is δ^{q_E}. The probability that it does not abort during the challenge step is $1 - \delta$. Therefore, the probability that $\mathcal{B}$ does not abort during the simulation is $\delta^{q_E}(1 - \delta)$. This value is maximized at $\delta_{opt} = 1 - 1/(q_E + 1)$. Using δ_{opt}, the probability that $\mathcal{B}$ does not abort is at least $1/e(1+q_E)$. This shows that $\mathcal{B}$'s advantage is at least $\epsilon/e(1+q_E)$ as required. □

The analysis used in the proof of Lemma 4.2 uses a similar technique to Coron's analysis of the Full Domain Hash signature scheme [9]. Next, we show that BasicPub is a semantically secure public key system if the BDH assumption holds.

Lemma 4.3. *Let H_2 be a random oracle from $\mathbb{G}_2$ to $\{0, 1\}^n$. Let $\mathcal{A}$ be an* IND-CPA *adversary that has advantage $\epsilon(k)$ against* BasicPub. *Suppose $\mathcal{A}$ makes a total of $q_{H_2} > 0$ queries to H_2. Then there is an algorithm $\mathcal{B}$ that solves the BDH problem for $\mathcal{G}$ with advantage at least $2\epsilon(k)/q_{H_2}$ and a running time $O(time(\mathcal{A}))$.*

Proof. Algorithm $\mathcal{B}$ is given as input the BDH parameters $\langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}\rangle$ produced by $\mathcal{G}$ and a random instance $\langle P, aP, bP, cP\rangle = \langle P, P_1, P_2, P_3\rangle$ of the BDH problem for these parameters, i.e. P is random in $\mathbb{G}_1^*$ and a, b, c are random in $\mathbb{Z}_q^*$ where q is the order of $\mathbb{G}_1, \mathbb{G}_2$. Let $D = \hat{e}(P, P)^{abc} \in \mathbb{G}_2$ be the solution to this BDH problem. Algorithm $\mathcal{B}$ finds D by interacting with $\mathcal{A}$ as follows:

Setup: Algorithm $\mathcal{B}$ creates the BasicPub public key $K_{pub} = \langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, n, P, P_{pub}, Q_{\mathsf{ID}}, H_2\rangle$ by setting $P_{pub} = P_1$ and $Q_{\mathsf{ID}} = P_2$. Here H_2 is a random oracle controlled by $\mathcal{B}$ as described below. Algorithm $\mathcal{B}$ gives $\mathcal{A}$ the BasicPub public key K_{pub}. Observe that the (unknown) private key associated to K_{pub} is $d_{\mathsf{ID}} = aQ_{\mathsf{ID}} = abP$.

H_2-queries: At any time algorithm $\mathcal{A}$ may issue queries to the random oracle H_2. To respond to these queries $\mathcal{B}$ maintains a list of tuples called the H_2^{list}. Each entry in the list is a tuple of the form $\langle X_j, H_j \rangle$. Initially the list is empty. To respond to query X_i algorithm $\mathcal{B}$ does the following:

1. If the query X_i already appears on the H_2^{list} in a tuple $\langle X_i, H_i \rangle$ then respond with $H_2(X_i) = H_i$.
2. Otherwise, $\mathcal{B}$ just picks a random string $H_i \in \{0,1\}^n$ and adds the tuple $\langle X_i, H_i \rangle$ to the H_2^{list}. It responds to $\mathcal{A}$ with $H_2(X_i) = H_i$.

Challenge: Algorithm $\mathcal{A}$ outputs two messages M_0, M_1 on which it wishes to be challenged. Algorithm $\mathcal{B}$ picks a random string $R \in \{0,1\}^n$ and defines C to be the ciphertext $C = \langle P_3, R \rangle$. Algorithm $\mathcal{B}$ gives C as the challenge to $\mathcal{A}$. Observe that, by definition, the decryption of C is $R \oplus H_2(\hat{e}(P_3, d_{\mathsf{ID}})) = R \oplus H_2(D)$.

Guess: Algorithm $\mathcal{A}$ outputs its guess $c' \in \{0,1\}$. At this point $\mathcal{B}$ picks a random tuple $\langle X_j, H_j \rangle$ from the H_2^{list} and outputs X_j as the solution to the given instance of BDH.

Algorithm $\mathcal{B}$ is simulating a real attack environment for algorithm $\mathcal{A}$ (it simulates the challenger and the oracle for H_2). We show that algorithm $\mathcal{B}$ outputs the correct answer D with probability at least $2\epsilon/q_{H_2}$ as required. The proof is based on comparing $\mathcal{A}$'s behavior in the simulation to its behavior in a real IND-CPA attack game (against a real challenger and a real random oracle for H_2).

Let $\mathcal{H}$ be the event that algorithm $\mathcal{A}$ issues a query for $H_2(D)$ at some point during the simulation above (this implies that at the end of the simulation D appears in some tuple on the H_2^{list}). We show that $\Pr[\mathcal{H}] \geq 2\epsilon$. This will prove that algorithm $\mathcal{B}$ outputs D with probability at least $2\epsilon/q_{H_2}$. We also study event $\mathcal{H}$ in the real attack game, namely the event that $\mathcal{A}$ issues a query for $H_2(D)$ when communicating with a real challenger and a real random oracle for H_2.

Claim 1: $\Pr[\mathcal{H}]$ in the simulation above is equal to $\Pr[\mathcal{H}]$ in the real attack.

Proof of claim. Let $\mathcal{H}_\ell$ be the event that $\mathcal{A}$ makes a query for $H_2(D)$ in one of its first ℓ queries to the H_2 oracle. We prove by induction on ℓ that $\Pr[\mathcal{H}_\ell]$ in the real attack is equal to $\Pr[\mathcal{H}_\ell]$ in the simulation for all $\ell \geq 0$. Clearly $\Pr[\mathcal{H}_0] = 0$ in both the simulation and in the real attack. Now suppose that for some $\ell > 0$ we have that $\Pr[\mathcal{H}_{\ell-1}]$ in the simulation is equal to $\Pr[\mathcal{H}_{\ell-1}]$ in the real attack. We show that the same holds for $\mathcal{H}_\ell$. We know that:

$$\begin{aligned} \Pr[\mathcal{H}_\ell] &= \Pr[\mathcal{H}_\ell \,|\, \mathcal{H}_{\ell-1}] \Pr[\mathcal{H}_{\ell-1}] + \Pr[\mathcal{H}_\ell \,|\, \neg\mathcal{H}_{\ell-1}] \Pr[\neg\mathcal{H}_{\ell-1}] \qquad (1) \\ &= \Pr[\mathcal{H}_{\ell-1}] + \Pr[\mathcal{H}_\ell \,|\, \neg\mathcal{H}_{\ell-1}] \Pr[\neg\mathcal{H}_{\ell-1}] \end{aligned}$$

We argue that $\Pr[\mathcal{H}_\ell \,|\, \neg\mathcal{H}_{\ell-1}]$ in the simulation is equal to $\Pr[\mathcal{H}_\ell \,|\, \neg\mathcal{H}_{\ell-1}]$ in the real attack. To see this observe that as long as $\mathcal{A}$ does not issue a query for $H_2(D)$ its view during the simulation is identical to its view in the real attack (against a real challenger and a real random oracle for H_2). Indeed, the public-key and the challenge are distributed as in the real attack. Similarly, all responses to H_2-queries are uniform and independent in $\{0,1\}^n$. Therefore, $\Pr[\mathcal{H}_\ell \,|\, \neg\mathcal{H}_{\ell-1}]$ in the simulation is equal to $\Pr[\mathcal{H}_\ell \,|\, \neg\mathcal{H}_{\ell-1}]$ in the real attack. It follows by (1) and the inductive hypothesis that $\Pr[\mathcal{H}_\ell]$ in the real attack is equal to $\Pr[\mathcal{H}_\ell]$ in the simulation. By induction on ℓ we obtain that $\Pr[\mathcal{H}]$ in the real attack is equal to $\Pr[\mathcal{H}]$ in the simulation. □

Claim 2: In the real attack we have $\Pr[\mathcal{H}] \geq 2\epsilon$.

Proof of claim. In the real attack, if $\mathcal{A}$ never issues a query for $H_2(D)$ then the decryption of C is independent of $\mathcal{A}$'s view (since $H_2(D)$ is independent of $\mathcal{A}$'s view). Therefore, in the real attack $\Pr[c = c' \mid \neg\mathcal{H}] = 1/2$. By definition of $\mathcal{A}$, we know that in the real attack $|\Pr[c = c'] - 1/2| \geq \epsilon$.

We show that these two facts imply that $\Pr[\mathcal{H}] \geq 2\epsilon$. To do so we first derive simple upper and lower bounds on $\Pr[c = c']$:

$$\begin{aligned}
\Pr[c = c'] &= \Pr[c = c' | \neg\mathcal{H}] \Pr[\neg\mathcal{H}] + \Pr[c = c' | \mathcal{H}] \Pr[\mathcal{H}] \leq \\
&\leq \Pr[c = c' | \neg\mathcal{H}] \Pr[\neg\mathcal{H}] + \Pr[\mathcal{H}] = \frac{1}{2} \Pr[\neg\mathcal{H}] + \Pr[\mathcal{H}] = \frac{1}{2} + \frac{1}{2} \Pr[\mathcal{H}] \\
\Pr[c = c'] &\geq \Pr[c = c' | \neg\mathcal{H}] \Pr[\neg\mathcal{H}] = \frac{1}{2} - \frac{1}{2} \Pr[\mathcal{H}]
\end{aligned}$$

It follows that $\epsilon \leq |\Pr[c = c'] - 1/2| \leq \frac{1}{2} \Pr[\mathcal{H}]$. Therefore, in the real attack $\Pr[\mathcal{H}] \geq 2\epsilon$. □

To complete the proof of Lemma 4.3 observe that by Claims 1 and 2 we know that $\Pr[\mathcal{H}] \geq 2\epsilon$ in the simulation above. Hence, at the end of the simulation, D appears in some tuple on the H_2^{list} with probability at least 2ϵ. It follows that $\mathcal{B}$ produces the correct answer with probability at least $2\epsilon/q_{H_2}$ as required. □

We note that one can slightly vary the reduction in the proof above to obtain different bounds. For example, in the 'Guess' step above one can avoid having to pick a random element from the H_2^{list} by using the random self reduction of the BDH problem. This requires running algorithm $\mathcal{A}$ multiple times (as in Theorem 7 of [42]). The success probability for solving the given BDH problem increases at the cost of also increasing the running time.

Proof of Theorem 4.1. The theorem follows directly from Lemma 4.2 and Lemma 4.3. Composing both reductions shows that an IND-ID-CPA adversary on BasicIdent with advantage $\epsilon(k)$ gives a BDH algorithm for $\mathcal{G}$ with advantage at least $2\epsilon(k)/e(1 + q_E)q_{H_2}$, as required. □

4.2 Identity-Based Encryption with Chosen Ciphertext Security

We use a technique due to Fujisaki-Okamoto [16] to convert the BasicIdent scheme of the previous section into a chosen ciphertext secure IBE system (in the sense of Section 2) in the random oracle model. Let $\mathcal{E}$ be a probabilistic public key encryption scheme. We denote by $\mathcal{E}_{pk}(M; r)$ the encryption of M using the random bits r under the public key pk. Fujisaki-Okamoto define the hybrid scheme $\mathcal{E}^{hy}$ as:

$$\mathcal{E}^{hy}_{pk}(M) = \langle\, \mathcal{E}_{pk}(\sigma; H_3(\sigma, M)),\ \ H_4(\sigma) \oplus M \,\rangle$$

Here σ is generated at random and H_3, H_4 are cryptographic hash functions. Fujisaki-Okamoto show that if $\mathcal{E}$ is a one-way encryption scheme then $\mathcal{E}^{hy}$ is a chosen ciphertext secure system (IND-CCA) in the random oracle model (assuming $\mathcal{E}_{pk}$ satisfies some natural constraints). We note that semantic security implies one-way encryption and hence the Fujisaki-Okamoto result also applies if $\mathcal{E}$ is semantically secure (IND-CPA).

We apply the Fujisaki-Okamoto transformation to BasicIdent and show that the resulting IBE system is IND-ID-CCA secure. We obtain the following IBE scheme which we call FullIdent. Recall that n is the length of the message to be encrypted.

Setup: As in the BasicIdent scheme. In addition, we pick a hash function $H_3 : \{0,1\}^n \times \{0,1\}^n \to \mathbb{Z}_q^*$, and a hash function $H_4 : \{0,1\}^n \to \{0,1\}^n$.

Extract: As in the BasicIdent scheme.

Encrypt: To encrypt $M \in \{0,1\}^n$ under the public key ID do the following: (1) compute $Q_{\mathsf{ID}} = H_1(\mathsf{ID}) \in \mathbb{G}_1^*$, (2) choose a random $\sigma \in \{0,1\}^n$, (3) set $r = H_3(\sigma, M)$, and (4) set the ciphertext to be

$$C = \langle rP,\ \ \sigma \oplus H_2(g_{\mathsf{ID}}^r),\ \ M \oplus H_4(\sigma)\rangle \quad \text{where} \quad g_{\mathsf{ID}} = \hat{e}(Q_{\mathsf{ID}}, P_{pub}) \in \mathbb{G}_2$$

Decrypt: Let $C = \langle U, V, W\rangle$ be a ciphertext encrypted using the public key ID. If $U \notin \mathbb{G}_1^*$ reject the ciphertext. To decrypt C using the private key $d_{\mathsf{ID}} \in \mathbb{G}_1^*$ do:

1. Compute $\quad V \oplus H_2(\hat{e}(d_{\mathsf{ID}}, U)) = \sigma$.
2. Compute $\quad W \oplus H_4(\sigma) = M$.
3. Set $r = H_3(\sigma, M)$. Test that $U = rP$. If not, reject the ciphertext.
4. Output M as the decryption of C.

This completes the description of FullIdent. Note that M is encrypted as $W = M \oplus H_4(\sigma)$. This can be replaced by $W = E_{H_4(\sigma)}(M)$ where E is a semantically secure symmetric encryption scheme (see [16]).

Security. The following theorem shows that FullIdent is a chosen ciphertext secure IBE (i.e. IND-ID-CCA), assuming BDH is hard in groups generated by $\mathcal{G}$.

Theorem 4.4. *Let the hash functions H_1, H_2, H_3, H_4 be random oracles. Then* FullIdent *is a chosen ciphertext secure IBE (*IND-ID-CCA*) assuming BDH is hard in groups generated by $\mathcal{G}$.*
Concretely, suppose there is an IND-ID-CCA *adversary $\mathcal{A}$ that has advantage $\epsilon(k)$ against the scheme* FullIdent *and $\mathcal{A}$ runs in time at most $t(k)$. Suppose $\mathcal{A}$ makes at most q_E extraction queries, at most q_D decryption queries, and at most $q_{H_2}, q_{H_3}, q_{H_4}$ queries to the hash functions H_2, H_3, H_4 respectively. Then there is a BDH algorithm $\mathcal{B}$ for $\mathcal{G}$ with running time $t_1(k)$ where:*

$$\begin{aligned} Adv_{\mathcal{G},\mathcal{B}}(k) &\geq 2FO_{adv}(\tfrac{\epsilon(k)}{e(1+q_E+q_D)},\ q_{H_4},\ q_{H_3},\ q_D)/q_{H_2} \\ t_1(k) &\leq FO_{time}(t(k), q_{H_4}, q_{H_3}) \end{aligned}$$

where the functions FO_{time} and FO_{adv} are defined in Theorem 4.5.

The proof of Theorem 4.4 is based on the following result of Fujisaki and Okamoto (Theorem 14 in [16]). Let BasicPub^{hy} be the result of applying the Fujisaki-Okamoto transformation to BasicPub.

Theorem 4.5 (Fujisaki-Okamoto). *Suppose $\mathcal{A}$ is an* IND-CCA *adversary that achieves advantage $\epsilon(k)$ when attacking* BasicPub^{hy}. *Suppose $\mathcal{A}$ has running time $t(k)$, makes at most q_D decryption queries, and makes at most q_{H_3}, q_{H_4} queries to the hash functions H_3, H_4 respectively. Then there is an* IND-CPA *adversary $\mathcal{B}$ against* BasicPub *with running time $t_1(k)$ and advantage $\epsilon_1(k)$ where*

$$\begin{aligned} \epsilon_1(k) &\geq FO_{adv}(\epsilon(k), q_{H_4}, q_{H_3}, q_D) = \frac{1}{2(q_{H_4}+q_{H_3})}\left[(\epsilon(k)+1)(1-2/q)^{q_D} - 1\right] \\ t_1(k) &\leq FO_{time}(t(k), q_{H_4}, q_{H_3}) = t(k) + O((q_{H_4}+q_{H_3})\cdot n), \quad \textit{and} \end{aligned}$$

Here q is the size of the groups $\mathbb{G}_1, \mathbb{G}_2$ and n is the length of σ.

In fact, Fujisaki-Okamoto prove a stronger result: Under the hypothesis of Theorem 4.5, BasicPub^{hy} would not even be a one-way encryption scheme. For our purposes the result in Theorem 4.5 is sufficient. To prove Theorem 4.4 we also need the following lemma to translate between an IND-ID-CCA chosen ciphertext attack on FullIdent and an IND-CCA chosen ciphertext attack on BasicPub^{hy}.

Lemma 4.6. *Let $\mathcal{A}$ be an* IND-ID-CCA *adversary that has advantage $\epsilon(k)$ against* FullIdent. *Suppose $\mathcal{A}$ makes at most $q_E > 0$ private key extraction queries and at most q_D decryption queries. Then there is an* IND-CCA *adversary $\mathcal{B}$ that has advantage at least $\frac{\epsilon(k)}{e(1+q_E+q_D)}$ against* BasicPub^{hy}. *Its running time is $O(time(\mathcal{A}))$.*

Proof. We construct an IND-CCA adversary $\mathcal{B}$ that uses $\mathcal{A}$ to gain advantage $\epsilon/e(1 + q_E + q_D)$ against $\mathsf{BasicPub}^{hy}$. The game between the challenger and the adversary $\mathcal{B}$ starts with the challenger first generating a random public key by running algorithm keygen of $\mathsf{BasicPub}^{hy}$. The result is a public key $K_{pub} = \langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, n, P, P_{pub}, Q_{\mathsf{ID}}, H_2, H_3, H_4\rangle$ and a private key $d_{\mathsf{ID}} = sQ_{\mathsf{ID}}$. The challenger gives K_{pub} to algorithm $\mathcal{B}$.

Algorithm $\mathcal{B}$ mounts an IND-CCA attack on the key K_{pub} using the help of algorithm $\mathcal{A}$. Algorithm $\mathcal{B}$ interacts with $\mathcal{A}$ as follows:

Setup: Same as in Lemma 4.2 (with H_3, H_4 included in the system parameters given to $\mathcal{A}$).

H_1-queries: These queries are handled as in Lemma 4.2.

Phase 1: Private key queries. Handled as in Lemma 4.2.

Phase 1: Decryption queries. Let $\langle \mathsf{ID}_i, C_i\rangle$ be a decryption query issued by algorithm $\mathcal{A}$. Let $C_i = \langle U_i, V_i, W_i\rangle$. Algorithm $\mathcal{B}$ responds to this query as follows:

1. Run the above algorithm for responding to H_1-queries to obtain a $Q_i \in \mathbb{G}_1^*$ such that $H_1(\mathsf{ID}_i) = Q_i$. Let $\langle \mathsf{ID}_i, Q_i, b_i, coin_i\rangle$ be the corresponding tuple on the H_1^{list}.
2. Suppose $coin_i = 0$. In this case run the algorithm for responding to private key queries to obtain the private key for the public key ID_i. Then use the private key to respond to the decryption query.
3. Suppose $coin_i = 1$. Then $Q_i = b_i Q_{\mathsf{ID}}$.
   - Recall that $U_i \in \mathbb{G}_1$. Set $C_i' = \langle b_i U_i,\ V_i,\ W_i\rangle$. Let $d_i = sQ_i$ be the (unknown) FullIdent private key corresponding to ID_i. Then the FullIdent decryption of C_i using d_i is the same as the $\mathsf{BasicPub}^{hy}$ decryption of C_i' using d_{ID}. To see this observe that:

$$\hat{e}(b_i U_i, d_{\mathsf{ID}}) = \hat{e}(b_i U_i, sQ_{\mathsf{ID}}) = \hat{e}(U_i, sb_i Q_{\mathsf{ID}}) = \hat{e}(U_i, sQ_i) = \hat{e}(U_i, d_i).$$

   - Relay the decryption query $\langle C_i'\rangle$ to the challenger and relay the challenger's response back to $\mathcal{A}$.

Challenge: Once algorithm $\mathcal{A}$ decides that Phase 1 is over it outputs a public key ID_{ch} and two messages M_0, M_1 on which it wishes to be challenged. Algorithm $\mathcal{B}$ responds as follows:

1. Algorithm $\mathcal{B}$ gives the challenger M_0, M_1 as the messages that it wishes to be challenged on. The challenger responds with a $\mathsf{BasicPub}^{hy}$ ciphertext $C = \langle U, V, W\rangle$ such that C is the encryption of M_c for a random $c \in \{0, 1\}$.
2. Next, $\mathcal{B}$ runs the algorithm for responding to H_1-queries to obtain a $Q \in \mathbb{G}_1^*$ such that $H_1(\mathsf{ID}_{ch}) = Q$. Let $\langle \mathsf{ID}_{ch}, Q, b, coin\rangle$ be the corresponding tuple on the H_1^{list}. If $coin = 0$ then $\mathcal{B}$ reports failure and terminates. The attack on $\mathsf{BasicPub}^{hy}$ failed.
3. We know $coin = 1$ and therefore $Q = bQ_{\mathsf{ID}}$. Recall that when $C = \langle U, V, W\rangle$ we have $U \in \mathbb{G}_1^*$. Set $C' = \langle b^{-1}U,\ V,\ W\rangle$, where b^{-1} is the inverse of b mod q. Algorithm $\mathcal{B}$ responds to $\mathcal{A}$ with the challenge C'. Note that, as in the proof of Lemma 4.2, C' is a FullIdent encryption of M_c under the public key ID_{ch} as required.

Phase 2: Private key queries. Algorithm $\mathcal{B}$ responds to private key extraction queries in the same way it did in Phase 1.

Phase 2: Decryption queries. Algorithm $\mathcal{B}$ responds to decryption queries in the same way it did in Phase 1. However, if the resulting decryption query relayed to the challenger is equal to the challenge ciphertext $C = \langle U, V, W\rangle$ then $\mathcal{B}$ reports failure and terminates. The attack on $\mathsf{BasicPub}^{hy}$ failed.

Guess: Eventually algorithm $\mathcal{A}$ outputs a guess c' for c. Algorithm $\mathcal{B}$ outputs c' as its guess for c.

Claim: If algorithm $\mathcal{B}$ does not abort during the simulation then algorithm $\mathcal{A}$'s view is identical to its view in the real attack. Furthermore, if $\mathcal{B}$ does not abort then $|\Pr[c = c'] - \frac{1}{2}| \geq \epsilon$. The probability is over the random bits used by $\mathcal{A}, \mathcal{B}$ and the challenger.

Proof of claim. The responses to H_1-queries are as in the real attack since each response is uniformly and independently distributed in $\mathbb{G}_1^*$. All responses to private key extraction queries and decryption queries are valid. Finally, the challenge ciphertext C' given to $\mathcal{A}$ is the $\mathsf{FullIdent}$ encryption of M_c for some random $c \in \{0, 1\}$. Therefore, by definition of algorithm $\mathcal{A}$ we have that $|\Pr[c = c'] - \frac{1}{2}| \geq \epsilon$. $\square$

It remains to bound the probability that algorithm $\mathcal{B}$ aborts during the simulation. The algorithm could abort for three reasons: (1) a bad private key query from $\mathcal{A}$ during phases 1 or 2, (2) $\mathcal{A}$ chooses a bad ID_{ch} to be challenged on, or (3) a bad decryption query from $\mathcal{A}$ during phase 2. We define three corresponding events:

$\mathcal{E}_1$ is the event that $\mathcal{A}$ issues a private key query during phase 1 or 2 that causes algorithm $\mathcal{B}$ to abort.

$\mathcal{E}_2$ is the event that $\mathcal{A}$ choose a public key ID_{ch} to be challenged on that causes algorithm $\mathcal{B}$ to abort.

$\mathcal{E}_3$ is the event that during phase 2 of the simulation Algorithm $\mathcal{A}$ issues a decryption query $\langle \mathsf{ID}_i, C_i \rangle$ so that the decryption query that $\mathcal{B}$ would relay to the $\mathsf{BasicPub}^{hy}$ challenger is equal to C. Recall that $C = \langle U, V, W \rangle$ is the challenge ciphertext from the $\mathsf{BasicPub}^{hy}$ challenger.

Claim: $\Pr[\neg\mathcal{E}_1 \wedge \neg\mathcal{E}_2 \wedge \neg\mathcal{E}_3] \geq \delta^{q_E+q_D}(1-\delta)$

Proof of claim. We prove the claim by induction on the maximum number of queries $q_E + q_D$ made by the adversary. Let $i = q_E + q_D$ and let $\mathcal{E}^{0\ldots i}$ be the event that $\mathcal{E}_1 \vee \mathcal{E}_3$ happens after $\mathcal{A}$ issues at most i queries. Similarly, let $\mathcal{E}^i$ be the event that $\mathcal{E}_1 \vee \mathcal{E}_3$ happens for the first time when $\mathcal{A}$ issues the i'th query. We prove by induction on i that $\Pr[\neg\mathcal{E}^{0\ldots i} \mid \neg\mathcal{E}_2] \geq \delta^i$. The claim follows because $\Pr[\neg\mathcal{E}_1 \wedge \neg\mathcal{E}_2 \wedge \neg\mathcal{E}_3] = \Pr[\neg\mathcal{E}_1 \wedge \neg\mathcal{E}_3 \mid \neg\mathcal{E}_2] \Pr[\neg\mathcal{E}_2] \geq \Pr[\neg\mathcal{E}_1 \wedge \neg\mathcal{E}_3 \mid \neg\mathcal{E}_2](1-\delta)$.

For $i = 0$ the claim is trivial since by definition $\Pr[\neg\mathcal{E}^{0\ldots 0}] = 1$. Now, suppose the claim holds for $i - 1$. Then

$$\begin{aligned}\Pr[\neg\mathcal{E}^{0\ldots i} \mid \neg\mathcal{E}_2] &= \Pr[\neg\mathcal{E}^{0\ldots i} \mid \neg\mathcal{E}^{0\ldots i-1} \wedge \neg\mathcal{E}_2] \Pr[\neg\mathcal{E}^{0\ldots i-1} \mid \neg\mathcal{E}_2] \\ &= \Pr[\neg\mathcal{E}^{i} \mid \neg\mathcal{E}^{0\ldots i-1} \wedge \neg\mathcal{E}_2] \Pr[\neg\mathcal{E}^{0\ldots i-1} \mid \neg\mathcal{E}_2] \geq \Pr[\neg\mathcal{E}^{i} \mid \neg\mathcal{E}^{0\ldots i-1} \wedge \neg\mathcal{E}_2]\delta^{i-1}\end{aligned}$$

Hence, it suffices to bound $q_i = \Pr[\neg\mathcal{E}^i \mid \neg\mathcal{E}^{0\ldots i-1} \wedge \neg\mathcal{E}_2]$. In other words, we bound the probability that the i'th query does not cause $\mathcal{E}^i$ to happen given that the first $i-1$ queries did not, and given that $\mathcal{E}_2$ does not occur. Consider the i'th query issued by $\mathcal{A}$ during the simulation. The query is either a private key query for $\langle \mathsf{ID}_i \rangle$ or a decryption query for $\langle \mathsf{ID}_i, C_i \rangle$ where $C_i = \langle U_i, V_i, W_i \rangle$. If the query is a decryption query we assume it takes place during phase 2 since otherwise it has no effect on $\mathcal{E}_3$.

Let $H_1(\mathsf{ID}_i) = Q_i$ and let $\langle \mathsf{ID}_i, Q_i, b_i, coin_i \rangle$ be the corresponding tuple on the H_1^{list}. Recall that when $coin_i = 0$ the query cannot cause event $\mathcal{E}_1$ to happen. Similarly, when $coin_i = 0$ the query cannot cause event $\mathcal{E}_3$ to happen since in this case $\mathcal{B}$ does not relay a decryption query to the $\mathsf{BasicPub}^{hy}$ challenger. We use these facts to bound q_i. There are four cases to consider. In the first three cases we assume ID_i is not equal to the public key ID_{ch} on which $\mathcal{A}$ is being challenged.

Case 1. The i'th query is the first time $\mathcal{A}$ issues a query containing ID_i. In this case $\Pr[coin_i = 0] = \delta$ and hence $q_i \geq \delta$.

Case 2. The public key ID_i appeared in a previous private key query. Since by assumption this earlier private key query did not cause $\mathcal{E}^{0\ldots i-1}$ to happen we know that $coin_i = 0$. Hence, we have $q_i = 1$.

Case 3. The public key ID_i appeared in a previous decryption query. Since by assumption this earlier decryption query did not cause event $\mathcal{E}^{0\ldots i-1}$ to happen we have that either $coin_i = 0$ or $coin_i$ is independent of $\mathcal{A}$'s current view. Either way we have that $q_i \geq \delta$.

Case 4. The public key ID_i is equal to the public key ID_{ch} on which $\mathcal{A}$ is being challenged. Then, by definition, the i'th query cannot be a private key query. Therefore, it must be a decryption query $\langle \mathsf{ID}_i, C_i \rangle$. Furthermore, since $\mathcal{E}_2$ did not happen we know that $coin_i = 1$ and hence $\mathcal{B}$ will relay a decryption query C_i' to the $\mathsf{BasicPub}^{hy}$ challenger. Let C' be the challenge ciphertext given to $\mathcal{A}$. By definition we know that $C_i \neq C'$. It follows that $C_i' \neq C$. Therefore this query cannot cause event $\mathcal{E}_3$ to happen. Hence, in this case $q_i = 1$.

To summarize, we see that whatever the i'th query is, we have that $q_i \geq \delta$. Therefore, we have that $\Pr[\neg\mathcal{E}^{0\ldots i} \mid \neg\mathcal{E}_2] \geq \delta^i$ as required. The claim now follows by setting $i = q_E + q_D$. □

To conclude the proof of Lemma 4.6 it remains to optimize the choice of δ. Since $\Pr[\neg\mathcal{E}_1 \wedge \neg\mathcal{E}_2 \wedge \neg\mathcal{E}_3] \geq \delta^{q_E+q_D}(1-\delta)$ the success probability is maximized at $\delta_{opt} = 1-1/(q_E+q_D+1)$. Using δ_{opt}, the probability that $\mathcal{B}$ does not abort is at least $\frac{1}{e(1+q_E+q_D)}$. This shows that $\mathcal{B}$'s advantage is at least $\epsilon/e(1+q_E+q_D)$ as required. □

Proof of Theorem 4.4. By Lemma 4.6 an IND-ID-CCA adversary on FullIdent implies an IND-CCA adversary on $\mathsf{BasicPub}^{hy}$. By Theorem 4.5 an IND-CCA adversary on $\mathsf{BasicPub}^{hy}$ implies an IND-CPA adversary on BasicPub. By Lemma 4.3 an IND-CPA adversary on BasicPub implies an algorithm for BDH. Composing all these reductions gives the required bounds. □

4.3 Relaxing the hashing requirements

Recall that the IBE system of Section 4.2 uses a hash function $H_1 : \{0,1\}^* \to \mathbb{G}_1^*$. The concrete IBE system presented in the next section uses $\mathbb{G}_1$ as a subgroup of the group of points on an elliptic curve. In practice, it is difficult to build hash functions that hash directly onto such groups. We therefore show how to relax the requirement of hashing directly onto $\mathbb{G}_1^*$. Rather than hash onto $\mathbb{G}_1^*$ we hash onto some set $A \subseteq \{0,1\}^*$ and then use a deterministic encoding function to map A onto $\mathbb{G}_1^*$.

Admissible encodings: Let $\mathbb{G}_1$ be a group and let $A \in \{0,1\}^*$ be a finite set. We say that an encoding function $L : A \to \mathbb{G}_1^*$ is *admissible* if it satisfies the following properties:

1. Computable: There is an efficient deterministic algorithm to compute $L(x)$ for any $x \in A$.

2. ℓ-to-1: For any $y \in \mathbb{G}_1^*$ the preimage of y under L has size exactly ℓ. In other words, $|L^{-1}(y)| = \ell$ for all $y \in \mathbb{G}_1^*$. Note that this implies that $|A| = \ell \cdot |\mathbb{G}_1^*|$.

3. Samplable: There is an efficient randomized algorithm $\mathcal{L}_S$ such that $\mathcal{L}_S(y)$ induces a uniform distribution on $L^{-1}(y)$ for any $y \in \mathbb{G}_1^*$. In other words, $\mathcal{L}_S(y)$ is a uniform random element in $L^{-1}(y)$.

We slightly modify FullIdent to obtain an IND-ID-CCA secure IBE system where H_1 is replaced by a hash function into some set A. Since the change is so minor we refer to this new scheme as FullIdent':

Setup: As in the FullIdent scheme. The only difference is that H_1 is replaced by a hash function $H_1' : \{0,1\}^* \to A$. The system parameters also include a description of an admissible encoding function $L : A \to \mathbb{G}_1^*$.

Extract, Encrypt: As in the FullIdent scheme. The only difference is that in Step 1 these algorithms compute $Q_{\mathsf{ID}} = L(H_1'(\mathsf{ID})) \in \mathbb{G}_1^*$.

Decrypt: As in the FullIdent scheme.

This completes the description of FullIdent'. The following theorem shows that FullIdent' is a chosen ciphertext secure IBE (i.e. IND-ID-CCA), assuming FullIdent is.

Theorem 4.7. *Let $\mathcal{A}$ be an* IND-ID-CCA *adversary on* FullIdent' *that achieves advantage $\epsilon(k)$. Suppose $\mathcal{A}$ makes at most q_{H_1} queries to the hash function H_1'. Then there is an* IND-ID-CCA *adversary $\mathcal{B}$ on* FullIdent *that achieves the same advantage $\epsilon(k)$ and time$(\mathcal{B})$ = time$(\mathcal{A})$ + $q_{H_1} \cdot$ time(L_S)*

Proof Sketch. Algorithm $\mathcal{B}$ attacks FullIdent by running algorithm $\mathcal{A}$. It relays all decryption queries, extraction queries, and hash queries from $\mathcal{A}$ directly to the challenger and relays the challenger's response back to $\mathcal{A}$. It only behaves differently when $\mathcal{A}$ issues a hash query to H_1'. Recall that $\mathcal{B}$ only has access to a hash function $H_1 : \{0,1\}^* \to \mathbb{G}_1^*$. To respond to H_1' queries algorithm $\mathcal{B}$ maintains a list of tuples $\langle \mathsf{ID}_j, y_j \rangle$ as explained below. We refer to this list as the $(H_1')^{list}$. The list is initially empty. When $\mathcal{A}$ queries the oracle H_1' at a point ID_i algorithm $\mathcal{B}$ responds as follows:

1. If the query ID_i already appears on the $(H_1')^{list}$ in a tuple $\langle \mathsf{ID}_i, y_i \rangle$, respond with $H_1'(\mathsf{ID}_i) = y_i \in A$.
2. Otherwise, $\mathcal{B}$ issues a query for $H_1(\mathsf{ID}_i)$. Say, $H_1(\mathsf{ID}_i) = \alpha \in \mathbb{G}_1^*$.
3. $\mathcal{B}$ runs the sampling algorithm $\mathcal{L}_S(\alpha)$ to generate a random element $y \in L^{-1}(\alpha)$.
4. $\mathcal{B}$ adds the tuple $\langle \mathsf{ID}_i, y \rangle$ to the $(H_1')^{list}$ and responds to $\mathcal{A}$ with $H_1'(\mathsf{ID}_i) = y \in A$. Note that y is uniformly distributed in A as required since α is uniformly distributed in $\mathbb{G}_1^*$ and L is an ℓ-to-1 map.

Algorithm $\mathcal{B}$'s responses to all of $\mathcal{A}$'s queries, including H_1' queries, are identical to $\mathcal{A}$'s view in the real attack. Hence, $\mathcal{B}$ will have the same advantage $\epsilon(k)$ in winning the game with the challenger. □

5 A concrete IBE system using the Weil pairing

In this section we use FullIdent' to describe a concrete IBE system based on the Weil pairing. We first review some properties of the pairing (see the Appendix for more details).

5.1 Properties of the Weil Pairing

Let p be a prime satisfying $p = 2 \bmod 3$ and let $q > 3$ be some prime factor of $p + 1$. Let E be the elliptic curve defined by the equation $y^2 = x^3 + 1$ over $\mathbb{F}_p$. We state a few elementary facts about this curve E (see [43] for more information). From here on we let $E(\mathbb{F}_{p^r})$ denote the group of points on E defined over $\mathbb{F}_{p^r}$.

Fact 1: Since $x^3 + 1$ is a permutation on $\mathbb{F}_p$ it follows that the group $E(\mathbb{F}_p)$ contains $p + 1$ points. We let O denote the point at infinity. Let $P \in E(\mathbb{F}_p)$ be a point of order q and let $\mathbb{G}_1$ be the subgroup of points generated by P.

Fact 2: For any $y_0 \in \mathbb{F}_p$ there is a unique point (x_0, y_0) on $E(\mathbb{F}_p)$, namely $x_0 = (y_0^2 - 1)^{1/3} \in \mathbb{F}_p$. Hence, if (x, y) is a random non-zero point on $E(\mathbb{F}_p)$ then y is uniform in $\mathbb{F}_p$. We use this property to build a simple admissible encoding function.

Fact 3: Let $1 \neq \zeta \in \mathbb{F}_{p^2}$ be a solution of $x^3 - 1 = 0$ in $\mathbb{F}_{p^2}$. Then the map $\phi(x, y) = (\zeta x, y)$ is an automorphism of the group of points on the curve E. Note that for any point $Q = (x, y) \in E(\mathbb{F}_p)$

we have that $\phi(Q) \in E(\mathbb{F}_{p^2})$, but $\phi(Q) \notin E(\mathbb{F}_p)$. Hence, $Q \in E(\mathbb{F}_p)$ is linearly independent of $\phi(Q) \in E(\mathbb{F}_{p^2})$.

Fact 4: Since the points $P \in \mathbb{G}_1$ and $\phi(P)$ are linearly independent they generate a group isomorphic to $\mathbb{Z}_q \times \mathbb{Z}_q$. We denote this group of points by $E[q]$.

Let $\mathbb{G}_2$ be the subgroup of $\mathbb{F}^*_{p^2}$ of order q. The Weil pairing on the curve $E(\mathbb{F}_{p^2})$ is a mapping $e : E[q] \times E[q] \to \mathbb{G}_2$ defined in the Appendix. For any $Q, R \in E(\mathbb{F}_p)$ the Weil pairing satisfies $e(Q, R) = 1$. In other words, the Weil pairing is degenerate on $E(\mathbb{F}_p)$, and hence degenerate on the group $\mathbb{G}_1$. To get a non-degenerate map we define the modified Weil pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ as follows:

$$\hat{e}(P, Q) = e(P, \phi(Q))$$

The modified Weil pairing satisfies the following properties:

1. Bilinear: For all $P, Q \in \mathbb{G}_1$ and for all $a, b \in \mathbb{Z}$ we have $\hat{e}(aP, bQ) = \hat{e}(P, Q)^{ab}$.
2. Non-degenerate: If P is a generator of $\mathbb{G}_1$ then $\hat{e}(P, P) \in \mathbb{F}^*_{p^2}$ is a generator of $\mathbb{G}_2$.
3. Computable: Given $P, Q \in \mathbb{G}_1$ there is an efficient algorithm, due to Miller, to compute $\hat{e}(P, Q) \in \mathbb{G}_2$. This algorithm is described in the Appendix. Its running time is comparable to exponentiation in $\mathbb{F}_p$.

Joux and Nguyen [28] point out that although the Computational Diffie-Hellman problem (CDH) appears to be hard in the group $\mathbb{G}_1$, the Decisional Diffie-Hellman problem (DDH) is easy in $\mathbb{G}_1$ (as discussed in Section 3).

BDH Parameter Generator $\mathcal{G}_1$: Given a security parameter $2 < k \in \mathbb{Z}$ the BDH parameter generator picks a random k-bit prime q and finds the smallest prime p such that (1) $p = 2 \bmod 3$, (2) q divides $p + 1$, and (3) q^2 does not divide $p + 1$. We write $p = \ell q + 1$. The group $\mathbb{G}_1$ is the subgroup of order q of the group of points on the curve $y^2 = x^3 + 1$ over $\mathbb{F}_p$. The group $\mathbb{G}_2$ is the subgroup of order q of $\mathbb{F}^*_{p^2}$. The bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ is the modified Weil pairing defined above.

The BDH parameter generator $\mathcal{G}_1$ is believed to satisfy the BDH assumption asymptotically. However, there is still the question of what values of p and q can be used in practice to make the BDH problem sufficiently hard. At the very least, we must ensure that the discrete log problem in $\mathbb{G}_1$ is sufficiently hard. As pointed out in Section 3 the discrete log problem in $\mathbb{G}_1$ is efficiently reducible to discrete log in $\mathbb{G}_2$ (see [32, 17]). Hence, computing discrete log in $\mathbb{F}^*_{p^2}$ is sufficient for computing discrete log in $\mathbb{G}_1$. In practice, for proper security of discrete log in $\mathbb{F}^*_{p^2}$ one often uses primes p that are at least 512-bits long (so that the group size is at least 1024-bits long). Consequently, one should not use this BDH parameter generator with primes p that are less than 512-bits long.

5.2 An admissible encoding function: MapToPoint

Let $\mathbb{G}_1, \mathbb{G}_2$ be two groups generated by $\mathcal{G}_1$ as defined above. Recall that the IBE system of Section 4.2 uses a hash function $H_1 : \{0,1\}^* \to \mathbb{G}_1^*$. By Theorem 4.7, it suffices to have a hash function $H_1 : \{0,1\}^* \to A$ for some set A, and an admissible encoding function $L : A \to \mathbb{G}_1^*$. In what follows the set A will be $\mathbb{F}_p$, and the admissible encoding function L will be called MapToPoint.

Let p be a prime satisfying $p = 2 \bmod 3$ and $p = \ell q - 1$ for some prime $q > 3$. We require that q does not divide ℓ (i.e. that q^2 does not divide $p+1$). Let E be the elliptic curve $y^2 = x^3 + 1$ over $\mathbb{F}_p$. Let $\mathbb{G}_1$ be the subgroup of points on E of order q. Suppose we already have a hash function $H_1 : \{0,1\}^* \to \mathbb{F}_p$.

Algorithm MapToPoint works as follows on input $y_0 \in \mathbb{F}_p$:

1. Compute $x_0 = (y_0^2 - 1)^{1/3} = (y_0^2 - 1)^{(2p-1)/3} \in \mathbb{F}_p$.
2. Let $Q = (x_0, y_0) \in E(\mathbb{F}_p)$ and set $Q_{\mathsf{ID}} = \ell Q \in \mathbb{G}_1$.
3. Output $\mathsf{MapToPoint}(y_0) = Q_{\mathsf{ID}}$.

This completes the description of MapToPoint.

We note that there are $\ell - 1$ values of $y_0 \in \mathbb{F}_p$ for which $\ell Q = \ell(x_0, y_0) = O$ (these are the non-O points of order dividing ℓ). Let $B \subset \mathbb{F}_p$ be the set of these y_0. When $H_1(\mathsf{ID})$ is one of these $\ell - 1$ values Q_{ID} is the identity element of $\mathbb{G}_1$. It is extremely unlikely for $H_1(\mathsf{ID})$ to hit one of these points – the probability is $1/q < 1/2^k$. Hence, for simplicity we say that $H_1(\mathsf{ID})$ only outputs elements in $\mathbb{F}_p \setminus B$, i.e. $H_1 : \{0,1\}^* \to \mathbb{F}_p \setminus B$. Algorithm MapToPoint can be easily extended to handle the values $y_0 \in B$ by hashing ID multiple times using different hash functions.

Lemma 5.1. $\mathsf{MapToPoint} : \mathbb{F}_p \setminus B \to \mathbb{G}_1^*$ *is an admissible encoding function.*

Proof. The map is clearly computable and is a $\ell - to - 1$ mapping. It remains to show that L is samplable. Let P be a generator of $E(\mathbb{F}_p)$. Given a $Q \in \mathbb{G}_1^*$ the sampling algorithm $\mathcal{L}_S$ does the following: (1) pick a random $b \in \{0, \ldots, \ell - 1\}$, (2) compute $Q' = \ell^{-1} \cdot Q + bqP = (x, y)$, and (3) output $\mathcal{L}_S(Q) = y \in \mathbb{F}_p$. Here ℓ^{-1} is the inverse of ℓ in $\mathbb{Z}_q^*$. This algorithm outputs a random element from the ℓ elements in $\mathsf{MapToPoint}^{-1}(Q)$ as required. □

5.3 A concrete IBE system

Using FullIdent' from Section 4.3 with the BDH parameter generator $\mathcal{G}_1$ and the admissible encoding function MapToPoint we obtain a concrete IBE system. Note that in this system, H_1 is a hash function from $\{0,1\}^*$ to $\mathbb{F}_p$ (where p is the finite field output by $\mathcal{G}_1$). The security of the system follows directly from Theorem 4.4 and Theorem 4.7. We summarize this in the following corollary.

Corollary 5.2. *The IBE system* FullIdent' *using the BDH parameter generator $\mathcal{G}_1$ and the admissible encoding* MapToPoint *is a chosen ciphertext secure IBE (i.e.* IND-ID-CCA *in the random oracle model) assuming $\mathcal{G}_1$ satisfies the BDH assumption.*

Performance. Algorithms Setup and Extract are very simple. At the heart of both algorithms is a standard multiplication on the curve $E(\mathbb{F}_p)$. Algorithm Encrypt requires that the encryptor compute the Weil pairing of Q_{ID} and P_{pub}. Note that this computation is independent of the message to be encrypted, and hence can be done once and for all. Once g_{ID} is computed the performance of the system is almost identical to standard ElGamal encryption. Decryption is a single Weil pairing computation. We note that the ciphertext length of BasicIdent using $\mathcal{G}_1$ is the same as in regular ElGamal encryption in $\mathbb{F}_p$.

6 Extensions and Observations

Tate pairing and other curves. Our IBE system works with any efficiently computable bilinear pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ between two groups $\mathbb{G}_1, \mathbb{G}_2$ as long as the BDH assumption holds. Many different curves, or more generally Abelian varieties, are believed to give rise to such maps. For example, one could use the curve $y^2 = x^3 + x$ over $\mathbb{F}_p$ with $p = 3 \bmod 4$ and its endomorphism $\phi : (x, y) \to (-x, iy)$ where $i^2 = -1$. As another example, Galbraith [18] suggests using supersingular

elliptic curves over a field of small characteristic to reduce the ciphertext size in our system. More general Abelian varieties are proposed by Rubin and Silverberg [39]. We note that both encryption and decryption in FullIdent can be made faster by using the Tate pairing on elliptic curves rather than the Weil pairing [19, 1].

Asymmetric pairings. Our IBE system can use slightly more general bilinear maps, namely maps of the form $\hat{e} : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_2$ where $\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_2$ are three groups of prime order q. Using the notation of Section 4.1 the only change to BasicIdent is that we take P and P_{pub} as elements in $\mathbb{G}_0$ and let H_1 be a hash function $H_1 : \{0,1\}^* \to \mathbb{G}_1^*$. Everything else remains the same. However, to make the proof of security go through (Lemma 4.2 in particular) we need a different complexity assumption which we call the co-BDH assumption: given random $P, aP, bP \in \mathbb{G}_0$ and $Q, aQ, cQ \in \mathbb{G}_1$ no polynomial time algorithm can compute $\hat{e}(P,Q)^{abc}$ with non-negligible probability. If one is willing to accept this assumption then we can avoid using supersingular curves and instead use elliptic curves over $\mathbb{F}_p$, $p > 3$ proposed by Miyaji et al. [35]. Curves $E/\mathbb{F}_p$ in this family are not supersingular and have the property that if q divides $|E(F_p)|$ then $E[q] \subseteq E(F_{p^6})$ (recall that $E[q]$ is the group containing all point in E of order dividing q). One way to use these curves is to set $\mathbb{G}_1$ to be a cyclic subgroup of $E(\mathbb{F}_p)$ of order q and $\mathbb{G}_0$ to be a different cyclic subgroup of $E(\mathbb{F}_{p^6})$ of the same order q. The standard Weil or Tate pairings on $\mathbb{G}_0 \times \mathbb{G}_1$ can be used as the bilinear map $\hat{e}$. Note that hashing public keys onto $\mathbb{G}_1 \subseteq E(\mathbb{F}_p)$ is easily done. Alternatively, to reduce the ciphertext size (which contains an element from $\mathbb{G}_0$) one could take $\mathbb{G}_0$ as a subgroup of order q of $E(\mathbb{F}_p)$ and $\mathbb{G}_1$ as a different subgroup of $E(\mathbb{F}_{p^6})$ of the same order. The question is how to hash public keys into $\mathbb{G}_1$. To do so, let $\mathrm{tr} : E(\mathbb{F}_{p^6}) \to E(\mathbb{F}_p)$ be the trace map on the curve and define $\mathbb{G}_1$ to be the subgroup of $E[q]$ containing all points P whose trace is O, i.e., $\mathrm{tr}(P) = O$. Then given a hash function $H : \{0,1\}^* \to E[q]$ we can hash a public key ID into $\mathbb{G}_1$ by computing: $H_1(\mathsf{ID}) = 6H(\mathsf{ID}) - \mathrm{tr}(H(\mathsf{ID})) \in \mathbb{G}_1$. Finally, we note that by modifying the security proof appropriately one can take $\mathbb{G}_1 = E[q]$ (a non-cyclic group) and then avoid computing traces while hashing into $\mathbb{G}_1$ (see also [18]).

Distributed PKG. In the standard use of an IBE in an e-mail system the master-key stored at the PKG must be protected in the same way that the private key of a CA is protected. One way of protecting this key is by distributing it among different sites using techniques of threshold cryptography [20]. Our IBE system supports this in a very efficient and robust way. Recall that the master-key is some $s \in \mathbb{Z}_q^*$. in order to generate a private key the PKG computes $Q_{priv} = sQ_{\mathsf{ID}}$, where Q_{ID} is derived from the user's public key ID. This can easily be distributed in a t-out-of-n fashion by giving each of the n PKGs one share s_i of a Shamir secret sharing of $s \bmod q$. When generating a private key each of the t chosen PKGs simply responds with $Q_{priv}^{(i)} = s_i Q_{\mathsf{ID}}$. The user can then construct Q_{priv} as $Q_{priv} = \sum \lambda_i Q_{priv}^{(i)}$ where the λ_i's are the appropriate Lagrange coefficients.

Furthermore, it is easy to make this scheme robust against dishonest PKGs using the fact that DDH is easy in $\mathbb{G}_1$. During setup each of the n PKGs publishes $P_{pub}^{(i)} = s_i P$. During a key generation request the user can verify that the response from the i'th PKG is valid by testing that:

$$\hat{e}(Q_{priv}^{(i)}, P) = \hat{e}(Q_{\mathsf{ID}}, P_{pub}^{(i)})$$

Thus, a misbehaving PKG will be immediately caught. There is no need for zero-knowledge proofs as in regular robust threshold schemes [21]. The PKG's master-key can be generated in a distributed fashion using the techniques of [22].

Note that a distributed master-key also enables threshold decryption on a *per-message* basis, without any need to derive the corresponding decryption key. For example, threshold decryption of BasicIdent ciphertext (U, V) is straightforward if each PKG responds with $\hat{e}(s_i Q_{\mathsf{ID}}, U)$.

Working in subgroups. The performance of our IBE system (Section 5) can be improved if we work in a small subgroup of the curve. For example, choose a 1024-bit prime $p = 2 \bmod 3$ with $p = aq - 1$ for some 160-bit prime q. The point P is then chosen to be a point of order q. Each public key ID is converted to a group point by hashing ID to a point Q on the curve and then multiplying the point by a. The system is secure if the BDH assumption holds in the group generated by P. The advantage is that the Weil computation is done on points of small order, and hence is much faster.

IBE implies signatures. Moni Naor has observed that an IBE scheme can be immediately converted into a public key signature scheme. The intuition is as follows. The private key for the signature scheme is the master key for the IBE scheme. The public key for the signature scheme is the global system parameters for the IBE scheme. The signature on a message M is the IBE decryption key for $\mathsf{ID} = M$. To verify a signature, choose a random message M', encrypt M' using the public key $\mathsf{ID} = M$, and then attempt to decrypt using the given signature on M as the decryption key. If the IBE scheme is IND-ID-CCA, then the signature scheme is existentially unforgeable against a chosen message attack. Note that, unlike most signature schemes, the signature verification algorithm here is randomized. This shows that secure IBE schemes incorporate both public key encryption and digital signatures. We note that the signature scheme derived from our IBE system has some interesting properties [6].

7 Escrow ElGamal encryption

In this section we show that the Weil pairing enables us to add a global escrow capability to the ElGamal encryption system. A single escrow key enables the decryption of ciphertexts encrypted under any public key. Paillier and Yung have shown how to add a global escrow capability to the Paillier encryption system [36]. Our ElGamal escrow system works as follows:

Setup: Let $\mathcal{G}$ be some BDH parameter generator. Given a security parameter $k \in \mathbb{Z}^+$, the algorithm works as follows:

Step 1: Run $\mathcal{G}$ on input k to generate a prime q, two groups $\mathbb{G}_1, \mathbb{G}_2$ of order q, and an admissible bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$. Choose a random generator P of $\mathbb{G}_1$.

Step 2: Pick a random $s \in \mathbb{Z}_q^*$ and set $Q = sP$.

Step 3: Choose a cryptographic hash function $H : \mathbb{G}_2 \to \{0,1\}^n$.

The message space is $\mathcal{M} = \{0,1\}^n$. The ciphertext space is $\mathcal{C} = \mathbb{G}_1 \times \{0,1\}^n$. The system parameters are $\mathsf{params} = \langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, n, P, Q, H \rangle$. The escrow key is $s \in \mathbb{Z}_q^*$.

keygen: A user generates a public/private key pair for herself by picking a random $x \in \mathbb{Z}_q^*$ and computing $P_{pub} = xP \in \mathbb{G}_1$. Her private key is x, her public key is P_{pub}.

Encrypt: To encrypt $M \in \{0,1\}^n$ under the public key P_{pub} do the following: (1) pick a random $r \in \mathbb{Z}_q^*$, and (2) set the ciphertext to be:

$$C = \langle rP,\ M \oplus H(g^r) \rangle \quad \text{where} \quad g = \hat{e}(P_{pub}, Q) \in \mathbb{G}_2$$

Decrypt: Let $C = \langle U, V \rangle$ be a ciphertext encrypted using P_{pub}. Then $U \in \mathbb{G}_1$. To decrypt C using the private key x do:

$$V \oplus H(\hat{e}(U, xQ)) = M$$

Escrow-decrypt: To decrypt $C = \langle U, V \rangle$ using the escrow key s do:

$$V \oplus H(\hat{e}(U, sP_{pub})) = M$$

A standard argument shows that assuming that BDH is hard for groups generated by $\mathcal{G}$ the system has semantic security in the random oracle model (recall that since DDH is easy we cannot prove semantic security based on DDH). Yet, the escrow agent can decrypt any ciphertext encrypted using any user's public key. The decryption capability of the escrow agent can be distributed using the PKG distribution techniques described in Section 6.

Using a similar hardness assumption, Verheul [46] described an ElGamal encryption system with non-global escrow. Each user constructs a public key with two corresponding private keys, and gives one of the private keys to the trusted third party. The trusted third party must maintain a database of all private keys given to it by the various users.

8 Summary and open problems

We defined chosen ciphertext security for identity-based systems and proposed a fully functional IBE system. The system has chosen ciphertext security in the random oracle model assuming BDH, a natural analogue of the computational Diffie-Hellman problem. The BDH assumption deserves further study considering the powerful cryptosystems derived from it. For example, it could be interesting to see whether the techniques of [30] can be used to prove that the BDH assumption is equivalent to the discrete log assumption on the curve for certain primes p.

Cocks [8] recently proposed another IBE system whose security is based on the difficulty of distinguishing quadratic residues from non-residues in the ring $\mathbb{Z}/N\mathbb{Z}$ where N is an RSA modulus (i.e., a product of two large primes). Cocks' system is somewhat harder to use in practice that the IBE system in this paper. Cocks' system uses bit-by-bit encryption and consequently outputs long ciphertexts. Also, encryption/decryption is a bit slower than the system described in this paper. Nevertheless, it is encouraging to see that IBE systems can be built using very different complexity assumptions.

It is an open problem to build chosen ciphertext secure identity based systems that are secure in the standard computation model (rather than the random oracle model). One might hope to use the techniques of Cramer-Shoup [10] to provide chosen ciphertext security based on DDH. Unfortunately, as mentioned in Section 3, the DDH assumption is false in the group of points on the curve E. However, simple variants of DDH do seem to hold. In particular, the following two distributions appear to be computationally indistinguishable: $\langle P, aP, bP, cP, abcP\rangle$ and $\langle P, aP, bP, cP, rP\rangle$ where a, b, c, r are random in $\mathbb{Z}_q$. We refer to this assumption as BDDH. A chosen ciphertext secure identity-based system strictly based on BDDH would be a plausible analogue of the Cramer-Shoup system. Building a chosen ciphertext secure IBE (IND-ID-CCA) in the standard model is currently an open problem.

Acknowledgments

The authors thank Moni Naor, Alice Silverberg, Ben Lynn, Steven Galbraith, Kenny Paterson, and Mike Scott for helpful discussions about this work.

References

[1] P. Barreto, H. Kim, B. Lynn, M. Scott, "Efficient Algorithms for Pairing-based Cryptosystems", in *Advances in Cryptology – Crypto 2002*, Lecture Notes in Computer Science, Springer-Verlag, 2002.

[2] M. Bellare, A. Desai, D. Pointcheval, P. Rogaway, "Relations among notions of security for public-key encryption schemes", in *Advances in Cryptology – Crypto '98*, Lecture Notes in Computer Science, Vol. 1462, Springer-Verlag, pp. 26–45, 1998.

[3] M. Bellare, P. Rogaway, "Random oracles are practical: a paradigm for designing efficient protocols", In ACM conference on Computers and Communication Security, pp. 62–73, 1993.

[4] D. Boneh, "The decision Diffie-Hellman problem", in *Proc. Third Algorithmic Number Theory Symposium*, Lecture Notes in Computer Science, Vol. 1423, Springer-Verlag, pp. 48–63, 1998.

[5] D. Boneh, M. Franklin, "Identity based encryption from the Weil pairing", extended abstract in *Advances in Cryptology – Crypto 2001*, Lecture Notes in Computer Science, Vol. 2139, Springer-Verlag, pp. 231–229, Aug. 2001. See also `http://eprint.iacr.org/2001/090/`

[6] D. Boneh, B. Lynn, H. Shacham, "Short signatures from the Weil pairing", in *Advances in Cryptology – AsiaCrypt 2001*, Lecture Notes in Computer Science, Vol. 2248, Springer-Verlag, pp. 514-532, 2001.

[7] M. Bellare, A. Boldyreva, S. Micali, "Public-key Encryption in a Multi-User Setting: Security Proofs and Improvements", in *Advances in Cryptology – Eurocrypt 2000*, Lecture Notes in Computer Science, Vol. 1807, Springer-Verlag, pp. 259–274, 2000.

[8] C. Cocks, "An identity based encryption scheme based on quadratic residues", Eighth IMA International Conference on Cryptography and Coding, Dec. 2001, Royal Agricultural College, Cirencester, UK.

[9] J. Coron, "On the exact security of Full-Domain-Hash", in *Advances in Cryptology – Crypto 2000*, Lecture Notes in Computer Science, Vol. 1880, Springer-Verlag, pp. 229–235, 2000.

[10] R. Cramer and V. Shoup, "A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack", in *Advances in Cryptology – Crypto '98*, Lecture Notes in Computer Science, Vol. 1462, Springer-Verlag, pp. 13–25, 1998.

[11] Y. Desmedt and J. Quisquater, "Public-key systems based on the difficulty of tampering", in *Advances in Cryptology – Crypto '86*, Lecture Notes in Computer Science, Vol. 263, Springer-Verlag, pp. 111–117, 1986.

[12] G. Di Crescenzo, R. Ostrovsky, and S. Rajagopalan, "Conditional Oblivious Transfer and Timed-Release Encryption", in *Advances in Cryptology – Eurocrypt '99*, Lecture Notes in Computer Science, Vol. 1592, pp. 74–89, 1999.

[13] D. Dolev, C. Dwork, M. Naor, "Non-malleable cryptography", *SIAM J. Computing*, Vol. 30(2), pp. 391–437, 2000.

[14] U. Feige, A. Fiat and A. Shamir, "Zero-knowledge proofs of identity", *J. Cryptology*, vol. 1, pp. 77–94, 1988.

[15] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems", in *Advances in Cryptology – Crypto '86*, Lecture Notes in Computer Science, Vol. 263, Springer-Verlag, pp. 186–194, 1986.

[16] E. Fujisaki and T. Okamoto, "Secure integration of asymmetric and symmetric encryption schemes", in *Advances in Cryptology – Crypto '99*, Lecture Notes in Computer Science, Vol. 1666, Springer-Verlag, pp. 537–554, 1999.

[17] G. Frey, M. Müller, H. Rück, "The Tate pairing and the discrete logarithm applied to elliptic curve cryptosystems", *IEEE Tran. on Info. Th.*, Vol. 45, pp. 1717–1718, 1999.

[18] S. Galbraith, "Supersingular curves in cryptography", in *Advances in Cryptology – AsiaCrypt 2001*, Lecture Notes in Computer Science, Vol. 2248, Springer-Verlag, pp. 495–513, 2001.

[19] S. Galbraith, K. Harrison, D. Soldera, "Implementing the Tate-pairing", in *Proc. Fifth Algorithmic Number Theory Symposium*, Lecture Notes in Computer Science, Springer-Verlag, 2002.

[20] P. Gemmell, "An introduction to threshold cryptography", in *CryptoBytes*, a technical newsletter of RSA Laboratories, Vol. 2, No. 7, 1997.

[21] R.Gennaro, S.Jarecki, H.Krawczyk, T.Rabin, "Robust and Efficient Sharing of RSA Functions", *J. Cryptology*, Vol. 13(2), pp. 273-300, 2000.

[22] R. Gennaro, S. Jarecki, H. Krawczyk, T. Rabin, "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems", *Advances in Cryptology – Eurocrypt '99*, Lecture Notes in Computer Science, Vol. 1592, Springer-Verlag, pp. 295–310, 1999.

[23] O. Goldreich, B. Pfitzmann and R. Rivest, "Self-delegation with controlled propagation -or- What if you lose your laptop", in *Advances in Cryptology – Crypto '98*, Lecture Notes in Computer Science, Vol. 1462, Springer-Verlag, pp. 153–168, 1998.

[24] S. Goldwasser, S. Micali, "Probabilistic Encryption", *J. Computer and System Sciences*, vol. 28, pp. 270-299, 1984.

[25] D. Hühnlein, M. Jacobson, D. Weber, "Towards Practical Non-interactive Public Key Cryptosystems Using Non-maximal Imaginary Quadratic Orders", in *Selected Areas in Cryptography*, Lecture Notes in Computer Science, Vol. 2012, Springer-Verlag, pp. 275–287, 2000.

[26] A. Joux, "A one round protocol for tripartite Diffie-Hellman", *Proc. Fourth Algorithmic Number Theory Symposium*, Lecture Notes in Computer Science, Vol. 1838, Springer-Verlag, pp. 385–394, 2000.

[27] A. Joux, "The Weil and Tate Pairings as Building Blocks for Public Key Cryptosystems", in *Proc. Fifth Algorithmic Number Theory Symposium*, Lecture Notes in Computer Science, Springer-Verlag, 2002.

[28] A. Joux, K. Nguyen, "Separating Decision Diffie-Hellman from Diffie-Hellman in cryptographic groups", J. Cryptology 16(4), pp. 239-247, 2003.

[29] S. Lang, *Elliptic functions*, Addison-Wesley, Reading, 1973.

[30] U. Maurer, "Towards proving the equivalence of breaking the Diffie-Hellman protocol and computing discrete logarithms", in *Advances in Cryptology – Crypto '94*, Lecture Notes in Computer Science, Vol. 839, pp. 271–281, 1994.

[31] U. Maurer and Y. Yacobi, "Non-interactive public-key cryptography", in *Advances in Cryptology – Crypto '91*, Lecture Notes in Computer Science, Vol. 547, Springer-Verlag, pp. 498–507, 1991.

[32] A. Menezes, T. Okamoto, S. Vanstone, "Reducing elliptic curve logarithms to logarithms in a finite field", *IEEE Tran. on Info. Th.*, Vol. 39, pp. 1639–1646, 1993.

[33] A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of applied cryptography*, CRC Press, Boca Raton, FL, 1996.

[34] V. Miller, "Short programs for functions on curves", unpublished manuscript.

[35] A. Miyaji, M. Nakabayashi, S. Takano, "New explicit condition of elliptic curve trace for FR-reduction", *IEICE Trans. Fundamentals*, Vol. E84 A, No. 5, May 2001.

[36] P. Paillier and M. Yung, "Self-escrowed public-key infrastructures" in *Information Security and Cryptology – ICISC '99*, Lecture Notes in Computer Science, Vol. 1787, Springer-Verlag, pp. 257–268, 1999.

[37] C. Rackoff, D. Simon, "Noninteractive zero-knowledge proof of knowledge and chosen ciphertext attack", in *Advances in Cryptology – Crypto '91*, Lecture Notes in Computer Science, Vol. 547, Springer-Verlag, pp. 433–444, 1991.

[38] R. Rivest, A. Shamir and D. Wagner, "Time lock puzzles and timed release cryptography," Technical report, MIT/LCS/TR-684

[39] K. Rubin, A. Silverberg, "Supersingular abelian varieties in cryptography", in *Advances in Cryptology – Crypto 2002*, Lecture Notes in Computer Science, Springer-Verlag, 2002.

[40] R. Sakai, K. Ohgishi, and M. Kasahara, "Cryptosystems based on pairings," In Proceedings of Symposium on Cryptography and Information Security, Japan, 2000.

[41] A. Shamir, "Identity-based cryptosystems and signature schemes", in *Advances in Cryptology – Crypto '84*, Lecture Notes in Computer Science, Vol. 196, Springer-Verlag, pp. 47–53, 1984.

[42] V. Shoup, 'Lower bounds for discrete logarithms and related problems', *In Proc. Eurocrypt '97*, Lect. Notes in Comp. Sci., Springer-Verlag, Berlin, **1233** (1997), 256–266.

[43] J. Silverman, *The arithmetic of elliptic curve*, Springer-Verlag, 1986.

[44] S. Tsuji and T. Itoh, "An ID-based cryptosystem based on the discrete logarithm problem", *IEEE Journal on Selected Areas in Communication*, vol. 7, no. 4, pp. 467–473, 1989.

[45] H. Tanaka, "A realization scheme for the identity-based cryptosystem", in *Advances in Cryptology – Crypto '87*, Lecture Notes in Computer Science, Vol. 293, Springer-Verlag, pp. 341–349, 1987.

[46] E. Verheul, "Evidence that XTR is more secure than supersingular elliptic curve cryptosystems", in *Advances in Cryptology – Eurocrypt 2001*, Lecture Notes in Computer Science, Vol. 2045, Springer-Verlag, pp. 195–210, 2001.

A Definition of the Weil pairing

We define the Weil pairing and show how to efficiently compute it using an algorithm due to Miller [34]. To be concrete we present the algorithm as it applies to supersingular elliptic curves defined over a prime field $\mathbb{F}_p$ with $p > 3$ (the curve $y^2 = x^3 + 1$ over $\mathbb{F}_p$ with $p = 2 \bmod 3$ is an example of such a curve). The definition and algorithm easily generalize to computing the Weil pairing over other elliptic curves. We state a few elementary facts about such curves [43]:

Fact 1: A supersingular curve $E/\mathbb{F}_p$ (with $p > 3$) contains $p + 1$ points in $\mathbb{F}_p$. We let O denote the point at infinity. The group of points over $\mathbb{F}_p$ forms a cyclic group of order $p + 1$. Let $P \in E(\mathbb{F}_p)$ be a point order n where n divides $p + 1$.

Fact 2: The group of points $E(\mathbb{F}_{p^2})$ contains a point Q of order n which is linearly independent of the points in $E(\mathbb{F}_p)$. Hence, $E(\mathbb{F}_{p^2})$ contains a subgroup which is isomorphic to the group $\mathbb{Z}_n^2$. The group is generated by $P \in E(\mathbb{F}_p)$ and $Q \in E(\mathbb{F}_{p^2})$. We denote this group by $E[n]$.

Throughout this section we let $\mathbb{G}_2$ denote the subgroup of $\mathbb{F}_{p^2}^*$ of order n. We will be working with the Weil pairing e which maps pairs of points in $E[n]$ to $\mathbb{G}_2$, i.e. $e : E[n] \times E[n] \to \mathbb{G}_2$. To define the pairing, we review a few basic concepts (see [29, pp. 243–245]). In what follows we let P and Q be arbitrary points in $E(\mathbb{F}_{p^2})$.

Divisors A divisor is a formal sum of points on the curve $E(\mathbb{F}_{p^2})$. We write divisors as $\mathcal{A} = \sum_P a_p(P)$ where $a_P \in \mathbb{Z}$ and $P \in E(\mathbb{F}_{p^2})$. For example, $\mathcal{A} = 3(P_1) - 2(P_2) - (P_3)$ is a divisor. We will only consider divisors $\mathcal{A} = \sum_P a_p(P)$ where $\sum_P a_p = 0$.

Functions Roughly speaking, a function f on the curve $E(\mathbb{F}_{p^2})$ can be viewed as a rational function $f(x, y) \in \mathbb{F}_{p^2}(x, y)$. For any point $P = (x, y) \in E(\mathbb{F}_{p^2})$ we define $f(P) = f(x, y)$.

Divisors of functions Let f be a function on the curve $E(\mathbb{F}_{p^2})$. We define its divisor, denoted by (f), as $(f) = \sum_P \mathrm{ord}_P(f) \cdot (P)$. Here $\mathrm{ord}_P(f)$ is the order of the zero that f has at the point P. For example, let $ax + by + c = 0$ be the line passing through the points $P_1, P_2 \in E(\mathbb{F}_{p^2})$ with $P_1 \neq \pm P_2$. This line intersects the curve at a third point $P_3 \in E(\mathbb{F}_{p^2})$. Then the function $f(x, y) = ax + by + c$ has three zeroes P_1, P_2, P_3 and a pole of order 3 at infinity. The divisor of f is $(f) = (P_1) + (P_2) + (P_3) - 3(O)$.

Principal divisors Let $\mathcal{A}$ be a divisor. If there exists a function f such that $(f) = \mathcal{A}$ then we say that $\mathcal{A}$ is a principal divisor. We know that a divisor $\mathcal{A} = \sum_P a_p(P)$ is principal if and only if $\sum_P a_p = 0$ and $\sum_P a_P P = O$. Note that the second summation is using the group action on the curve. Furthermore, given a principal divisor $\mathcal{A}$ there exists a *unique* function f (up to constant multiples) such that $(A) = (f)$.

Equivalence of divisors We say that two divisors $\mathcal{A}, \mathcal{B}$ are equivalent if their difference $\mathcal{A} - \mathcal{B}$ is a principal divisor. We know that any divisor $\mathcal{A} = \sum_P a_p(P)$ (with $\sum_P a_P = 0$) is equivalent to a divisor of the form $\mathcal{A}' = (Q) - (O)$ for some $Q \in E$. Observe that $Q = \sum_P a_P P$.

Notation Given a function f and a divisor $\mathcal{A} = \sum_P a_p(P)$ we define $f(\mathcal{A})$ as $f(\mathcal{A}) = \prod_P f(P)^{a_P}$. Note that since $\sum_P a_P = 0$ we have that $f(\mathcal{A})$ remains unchanged if instead of f we use cf for any $c \in \mathbb{F}_{p^2}$.

We are now ready to define the Weil pairing of two points $P, Q \in E[n]$. Let $\mathcal{A}_P$ be some divisor equivalent to the divisor $(P) - (O)$. We know that $n\mathcal{A}_P$ is a principal divisor (it is equivalent to

$n(P) - n(O)$ which is clearly a principal divisor). Hence, there exists a function f_P such that $(f_P) = n\mathcal{A}_P$. Define $\mathcal{A}_Q$ and f_Q analogously. The Weil pairing of P and Q is defined as:

$$e(P,Q) = \frac{f_P(\mathcal{A}_Q)}{f_Q(\mathcal{A}_P)}$$

This ratio defines the Weil pairing of P and Q whenever it is well defined (no division by zero occurred). If this ratio is undefined we use different divisors $\mathcal{A}_P, \mathcal{A}_Q$ to define $e(P,Q)$.

We briefly show that the Weil pairing is well defined. That is, the value of $e(P,Q)$ is independent of the choice of the divisor $\mathcal{A}_P$ as long as $\mathcal{A}_P$ is equivalent to $(P)-(O)$ and $\mathcal{A}_P$ leads to a well defined value. The same holds for $\mathcal{A}_Q$. Let $\hat{\mathcal{A}}_P$ be a divisor equivalent to $\mathcal{A}_P$ and let $\hat{f}_P$ be a function so that $(\hat{f}_P) = n\hat{\mathcal{A}}_P$. Then $\hat{\mathcal{A}}_P = \mathcal{A}_P + (g)$ for some function g and $\hat{f}_P = f_P \cdot g^n$. We have that:

$$e(P,Q) = \frac{\hat{f}_P(\mathcal{A}_Q)}{f_Q(\hat{\mathcal{A}}_P)} = \frac{f_P(\mathcal{A}_Q)g(\mathcal{A}_Q)^n}{f_Q(\mathcal{A}_P)f_Q((g))} = \frac{f_P(\mathcal{A}_Q)}{f_Q(\mathcal{A}_P)} \cdot \frac{g(n\mathcal{A}_Q)}{f_Q((g))} = \frac{f_P(\mathcal{A}_Q)}{f_Q(\mathcal{A}_P)} \cdot \frac{g((f_Q))}{f_Q((g))} = \frac{f_P(\mathcal{A}_Q)}{f_Q(\mathcal{A}_P)}$$

The last equality follows from the following fact known as Weil reciprocity: for any two functions f, g we have that $f(\,(g)\,) = g(\,(f)\,)$. Hence, the Weil pairing is well defined.

Fact A.1. *The Weil pairing has the following properties for points in $E[n]$:*

- *For all $P \in E[n]$ we have: $e(P,P) = 1$.*
- *Bilinear: $e(P_1 + P_2, Q) = e(P_1, Q) \cdot e(P_2, Q)$ and $e(P, Q_1 + Q_2) = e(P, Q_1) \cdot e(P, Q_2)$.*
- *When $P, Q \in E[n]$ are collinear then $e(P,Q) = 1$. Similarly, $e(P,Q) = e(Q,P)^{-1}$.*
- *n'th root: for all $P, Q \in E[n]$ we have $e(P,Q)^n = 1$, i.e. $e(P,Q) \in \mathbb{G}_2$.*
- *Non-degenerate in the following sense: if $P \in E[n]$ satisfies $e(P,Q) = 1$ for all $Q \in E[n]$ then $P = O$.*

As discussed in Section 5, our concrete IBE scheme uses the modified Weil pairing $\hat{e}(P,Q) = e(P, \phi(Q))$, where ϕ is an automorphism on the group of points of E.

Tate pairing. The Tate pairing [17] is another bilinear pairing that has the required properties for our system. We slightly modify the original definition to fit our purpose. Define the Tate pairing of two points $P, Q \in E[n]$ as $T(P,Q) = f_P(\mathcal{A}_Q)^{|\mathbb{F}_{p^2}^*|/n}$ where f_P and $\mathcal{A}_Q$ are defined as above. This definition gives a computable bilinear pairing $T : E[n] \times E[n] \to \mathbb{G}_2$.

B Computing the Weil pairing

Given two points $P, Q \in E[n]$ we show how to compute $e(P,Q) \in \mathbb{F}_{p^2}^*$ using $O(\log p)$ arithmetic operations in $\mathbb{F}_p$. We assume $P \neq Q$. We proceed as follows: pick two random points $R_1, R_2 \in E[n]$. Consider the divisors $\mathcal{A}_P = (P + R_1) - (R_1)$ and $\mathcal{A}_Q = (Q + R_2) - (R_2)$. These divisors are equivalent to $(P) - (O)$ and $(Q) - (O)$ respectively. Hence, we can use $\mathcal{A}_P$ and $\mathcal{A}_Q$ to compute the Weil pairing as:

$$e(P,Q) = \frac{f_P(\mathcal{A}_Q)}{f_Q(\mathcal{A}_P)} = \frac{f_P(Q + R_2)f_Q(R_1)}{f_P(R_2)f_Q(P + R_1)}$$

This expression is well defined with very high probability over the choice of R_1, R_2 (the probability of failure is at most $O(\frac{\log p}{p})$). In the rare event that a division by zero occurs during the computation of $e(P,Q)$ we simply pick new random points R_1, R_2 and repeat the process.

To evaluate $e(P,Q)$ it suffices to show how to evaluate the function f_P at $\mathcal{A}_Q$. Evaluating $f_Q(\mathcal{A}_P)$ is done analogously. We evaluate $f_P(\mathcal{A}_Q)$ using repeated doubling. For a positive integer b define the divisor

$$\mathcal{A}_b = b(P+R_1) - b(R_1) - (bP) + (O)$$

It is a principal divisor and therefore there exists a function f_b such that $(f_b) = \mathcal{A}_b$. Observe that $(f_P) = (f_n)$ and hence, $f_P(\mathcal{A}_Q) = f_n(\mathcal{A}_Q)$. It suffices to show how to evaluate $f_n(\mathcal{A}_Q)$.

Lemma B.1. *There is an algorithm $\mathcal{D}$ that given $f_b(\mathcal{A}_Q), f_c(\mathcal{A}_Q)$ and $bP, cP, (b+c)P$ for some $b, c > 0$ outputs $f_{b+c}(\mathcal{A}_Q)$. The algorithm only uses a (small) constant number of arithmetic operations in $\mathbb{F}_{p^2}$.*

Proof. We first define two auxiliary linear functions g_1, g_2:

1. Let $a_1x + b_1y + c_1 = 0$ be the line passing through the points bP and cP (if $b = c$ then let $a_1x + b_1y + c_1 = 0$ be the line tangent to E at bP). Define $g_1(x,y) = a_1x + b_1y + c_1$.
2. Let $x + c_2 = 0$ be the vertical line passing through the point $(b+c)P$. Define $g_2(x,y) = x + c_2$

The divisors of these functions are:

$$\begin{aligned}(g_1) &= (bP) + (cP) + (-(b+c)P) - 3(O)\\ (g_2) &= ((b+c)P) + (-(b+c)P) - 2(O)\end{aligned}$$

By definition we have that:

$$\begin{aligned}\mathcal{A}_b &= b(P+R_1) - b(R_1) - (bP) + (O)\\ \mathcal{A}_c &= c(P+R_1) - c(R_1) - (cP) + (O)\\ \mathcal{A}_{b+c} &= (b+c)(P+R_1) - (b+c)(R_1) - ((b+c)P) + (O)\end{aligned}$$

It now follows that: $\mathcal{A}_{b+c} = \mathcal{A}_b + \mathcal{A}_c + (g_1) - (g_2)$. Hence:

$$f_{b+c}(\mathcal{A}_Q) = f_b(\mathcal{A}_Q) \cdot f_c(\mathcal{A}_Q) \cdot \frac{g_1(\mathcal{A}_Q)}{g_2(\mathcal{A}_Q)} \tag{2}$$

This shows that to evaluate $f_{b+c}(\mathcal{A}_Q)$ it suffices to evaluate $g_i(\mathcal{A}_Q)$ for all $i = 1, 2$ and plug the results into equation 2. Hence, given $f_b(\mathcal{A}_Q), f_c(\mathcal{A}_Q)$ and $bP, cP, (b+c)P$ one can compute $f_{b+c}(\mathcal{A}_Q)$ using a constant number of arithmetic operations. □

Let $\mathcal{D}\big(f_b(\mathcal{A}_Q), f_c(\mathcal{A}_Q), bP, cP, (b+c)P\big) = f_{b+c}(\mathcal{A}_Q)$ denote the output of Algorithm $\mathcal{D}$ of Lemma B.1 above. Then one can compute $f_P(\mathcal{A}_Q) = f_n(\mathcal{A}_Q)$ using the following standard repeated doubling procedure. Let $n = b_m b_{m-1} \ldots b_1 b_0$ be the binary representation of n, i.e. $n = \sum_{i=0}^{m} b_i 2^i$.

Init: Set $Z = O$, $V = f_0(\mathcal{A}_Q) = 1$, and $k = 0$.

Iterate: For $i = m, m-1, \ldots, 1, 0$ do:

1: If $b_i = 1$ then do: Set $V = \mathcal{D}(V, f_1(\mathcal{A}_Q), Z, P, Z+P)$, set $Z = Z + P$, and set $k = k + 1$.
2: If $i > 0$ set $V = \mathcal{D}(V, V, Z, Z, 2Z)$, set $Z = 2Z$, and set $k = 2k$.

3: Observe that at the end of each iteration we have $Z = kP$ and $V = f_k(\mathcal{A}_Q)$.

Output: After the last iteration we have $k = n$ and therefore $V = f_n(\mathcal{A}_Q)$ as required.

To evaluate the Weil pairing $e(P, Q)$ we run the above algorithm once to compute $f_P(\mathcal{A}_Q)$ and once to compute $f_Q(\mathcal{A}_P)$. The Tate pairing is evaluated similarly. Note that the repeated squaring algorithm needs to evaluate $f_1(\mathcal{A}_Q)$. This is easily done since the function $f_1(x, y)$ (whose divisor is $(f_1) = (P + R_1) - (R_1) - (P) + (O)$) can be written out explicitly as follows:

1. Let $a_1x + b_1y + c_1 = 0$ be the line passing through the points P and R_1. Define the function: $g_1(x, y) = a_1x + b_1y + c_1$.
2. Let $x + c_2 = 0$ be the vertical line passing through the point $P + R_1$. Define the function: $g_2(x, y) = x + c_2$.
3. The function $f_1(x, y)$ is simply $f_1(x, y) = g_2(x, y)/g_1(x, y)$ which is easy to evaluate in $\mathbb{F}_{p^2}$.

Aggregate and Verifiably Encrypted Signatures from Bilinear Maps

Dan Boneh
dabo@cs.stanford.edu

Craig Gentry
cgentry@docomolabs-usa.com

Ben Lynn
blynn@cs.stanford.edu

Hovav Shacham
hovav@cs.stanford.edu

Abstract

An aggregate signature scheme is a digital signature that supports aggregation: Given n signatures on n distinct messages from n distinct users, it is possible to aggregate all these signatures into a single short signature. This single signature (and the n original messages) will convince the verifier that the n users did indeed sign the n original messages (i.e., user i signed message M_i for $i = 1, \ldots, n$). In this paper we introduce the concept of an aggregate signature, present security models for such signatures, and give several applications for aggregate signatures. We construct an efficient aggregate signature from a recent short signature scheme based on bilinear maps due to Boneh, Lynn, and Shacham. Aggregate signatures are useful for reducing the size of certificate chains (by aggregating all signatures in the chain) and for reducing message size in secure routing protocols such as SBGP. We also show that aggregate signatures give rise to verifiably encrypted signatures. Such signatures enable the verifier to test that a given ciphertext C is the encryption of a signature on a given message M. Verifiably encrypted signatures are used in contract-signing protocols. Finally, we show that similar ideas can be used to extend the short signature scheme to give simple ring signatures.

1 Introduction

Many real-world applications involve signatures on many different messages generated by many different users. For example, in a Public Key Infrastructure (PKI) of depth n, each user is given a chain of n certificates. The chain contains n signatures by n Certificate Authorities (CAs) on n distinct certificates. Similarly, in the Secure BGP protocol (SBGP) [18] each router receives a list of n signatures attesting to a certain path of length n in the network. A router signs its own segment in the path and forwards the resulting list of $n + 1$ signatures to the next router. As a result, the number of signatures in routing messages is linear in the length of the path. Both applications would benefit from a method for compressing the list of signatures on distinct messages issued by distinct parties. Specifically, X.509 certificate chains could be shortened by compressing the n signatures in the chain into a single signature.

An aggregate signature scheme enables us to achieve precisely this type of compression. Suppose each of n users has a public-private key pair (PK_i, SK_i). User u_i signs message M_i to obtain a signature σ_i. Then there is a public aggregation algorithm that takes as input all of $\sigma_1, \ldots, \sigma_n$ and outputs a short compressed signature σ. Anyone can aggregate the n signatures. Moreover, the aggregation can be performed incrementally. That is, signatures σ_1, σ_2 can be aggregated into σ_{12} which can then be further aggregated with σ_3 to obtain σ_{123}. When aggregating signatures in a certificate chain, each CA can incrementally aggregate its own signature into the chain. There is also an aggregate verification algorithm that takes $PK_1, \ldots, PK_n$, $M_1, \ldots, M_n$, and σ and decides

whether the aggregate signature is valid. Intuitively, the security requirement is that the aggregate signature σ is declared valid only if the aggregator who created σ was given all of $\sigma_1, \ldots, \sigma_n$. Precise security definitions are given in Sect. 3.2. Thus, an aggregate signature provides non-repudiation at once on many different messages by many users.

We construct an aggregate signature scheme based on a recent short signature due to Boneh, Lynn, and Shacham (BLS) [6]. This signature scheme works in any group where the Decision Diffie-Hellman problem (DDH) is easy, but the Computational Diffie-Hellman problem (CDH) is hard. We refer to such groups as gap groups [6, 26]. Recently there have been a number of constructions using such gap groups [6, 19, 8, 4]. Surprisingly, general gap groups are insufficient for constructing efficient aggregate signatures. Instead, our construction uses a pair of groups G_1, G_T and a bilinear map $e : G_1 \times G_1 \to G_T$ where CDH is hard in G_1. Joux and Nguyen [17] showed that the map e can be used to solve DDH in G_1, and so G_1 is a gap group. It is the extra structure provided by the bilinear map that enables us to construct an efficient aggregate signature scheme. We do not know how to build efficient aggregate signatures from general gap groups. Thus, our construction is an example where the bilinear map provides extra functionality beyond a simple algorithm for solving DDH. Bilinear maps were previously used for three-way Diffie-Hellman [16], Identity-Based Encryption (IBE) [5], and Hierarchical IBE [15, 13].

Aggregate signatures are related to multisignatures [20, 25, 24, 4]. In multisignatures, a set of users all sign the *same message* and the result is a single signature. Recently, Micali et al. [20] defined a security model for multisignatures and gave some constructions and applications. Multisignatures are insufficient for the applications we have in mind, such as certificate chains and SBGP. For these applications we must be able to aggregate signatures on distinct messages. We note that recently Boldyreva [4] showed that general gap groups are sufficient for constructing multisignatures from BLS signatures. As noted above, to obtain aggregate signatures, one needs the extra structure provided by bilinear maps.

Our application of aggregate signatures to compressing certificate chains is related to an open problem posed by Micali and Rivest [21]: Given a certificate chain and some special additional signatures, can intermediate links in the chain be cut out? Aggregate signatures allow the compression of certificate chains without any additional signatures, but a verifier must still be aware of all intermediate links in the chain. We note that batch RSA [9] also provides some signature compression, but only for signatures produced by a single signer.

As a further application for aggregate signatures we show in Sect. 4 that certain aggregate signature schemes give rise to simple verifiably encrypted signatures. These signatures enable user Alice to give Bob a signature on a message M encrypted using a third party's public key and Bob to verify that the encrypted signature is valid. Verifiably encrypted signatures are used in optimistic contract signing protocols [1, 2] to enable fair exchange. Previous constructions [1, 27] require zero knowledge proofs to verify an encrypted signature. The verifiably encrypted signatures in Section 4 are short and can be validated efficiently. We note that the resulting contract signing protocol is not abuse-free in the sense of [10].

As a third application of these ideas we construct in Sect. 5 a simple ring signature [28] using bilinear maps. As above, the construction using a bilinear map is simpler and more efficient than constructions that only make use of gap groups.

2 Signature Schemes Based on Co-Gap Diffie-Hellman

We first review a few concepts related to bilinear maps and Gap Diffie-Hellman signatures [6]. Throughout the paper we use the following notation:

1. G_1 and G_2 are two (multiplicative) cyclic groups of prime order p;
2. g_1 is a generator of G_1 and g_2 is a generator of G_2;
3. ψ is a computable isomorphism from G_2 to G_1, with $\psi(g_2) = g_1$; and
4. e is a computable bilinear map $e : G_1 \times G_2 \to G_T$ as described below.

The isomorphism ψ is mostly needed for the proofs of security. To keep the discussion general, we simply assume that ψ exists and is efficiently computable. When G_1, G_2 are subgroups of the group of points of an elliptic curve $E/\mathbb{F}_q$, the trace map on the curve can be used as this isomorphism (we assume $G_1 \subseteq E(\mathbb{F}_q)$ and $G_2 \subseteq E(\mathbb{F}_{q^r})$).

Throughout the paper, we consider bilinear maps $e : G_1 \times G_2 \to G_T$ where all groups G_1, G_2, G_T are multiplicative and of prime order p. One could set $G_1 = G_2$. However, we allow for the more general case where $G_1 \neq G_2$ so that our constructions can make use of certain families of non-supersingular elliptic curves defined by Miyaji et al. [22]. These curves give rise to very short signatures [6]. This will lead in turn to short aggregate signatures, ring signatures, etc. To handle the case $G_1 \neq G_2$ we define the co-CDH and co-DDH problems [6]. When $G_1 = G_2$, these problems reduce to the standard CDH and DDH problems. Hence, for the remainder of the paper, although we handle arbitrary G_1, G_2, for simplicity, the reader may assume $G_1 = G_2, g_1 = g_2$, and $\psi = I$, the identity map.
With this setup we obtain natural generalizations of the CDH and DDH problems:

Computational Co-Diffie-Hellman. Given $g_2, g_2^a \in G_2$ and $h \in G_1$ compute $h^a \in G_1$.

Decision Co-Diffie-Hellman. Given $g_2, g_2^a \in G_2$ and $h, h^b \in G_1$ output `yes` if $a = b$ and `no` otherwise. When the answer is `yes` we say that (g_2, g_2^a, h, h^a) is a co-Diffie-Hellman tuple.

When $G_1 = G_2$ and $g_1 = g_2$, these problems reduce to the standard CDH and DDH. Next we define co-GDH gap groups to be group pairs G_1 and G_2 on which co-DDH is easy but co-CDH is hard.

Definition 2.1. Two groups (G_1, G_2) are a decision group pair for co-Diffie-Hellman if the group action on G_1, the group action on G_2, and the map ψ from G_2 to G_1 can be computed in one time unit, and Decision co-Diffie-Hellman on (G_1, G_2) can be solved in one time unit.

Definition 2.2. The advantage of an algorithm $\mathcal{A}$ in solving the Computational co-Diffie-Hellman problem in groups G_1 and G_2 is

$$\mathsf{Adv\ co\text{-}CDH}_{\mathcal{A}} \stackrel{\text{def}}{=} \Pr\left[\mathcal{A}(g_2, g_2^a, h) = h^a : a \stackrel{\text{R}}{\leftarrow} \mathbb{Z}_p, h \stackrel{\text{R}}{\leftarrow} G_1\right] .$$

The probability is taken over the choice of a, h, and $\mathcal{A}$'s coin tosses. An algorithm $\mathcal{A}$ (t, ϵ)-breaks Computational co-Diffie-Hellman on G_1 and G_2 if $\mathcal{A}$ runs in time at most t, and $\mathsf{Adv\ co\text{-}CDH}_{\mathcal{A}}$ is at least ϵ. Two Groups (G_1, G_2) are a (t, ϵ)-co-GDH group pair if they are a decision group pair for co-Diffie-Hellman and no algorithm (t, ϵ)-breaks Computational co-Diffie-Hellman on them.

2.1 Bilinear Maps

Let G_1 and G_2 be two groups as above, with an additional group G_T such that $|G_1| = |G_2| = |G_T|$. A bilinear map is a map $e : G_1 \times G_2 \to G_T$ with the following properties:

1. Bilinear: for all $u \in G_1, v \in G_2$ and $a, b \in \mathbb{Z}$, $e(u^a, v^b) = e(u, v)^{ab}$.

2. Non-degenerate: $e(g_1, g_2) \neq 1$.

These properties imply two more: for any $u_1, u_2 \in G_1, v \in G_2$, $e(u_1 u_2, v) = e(u_1, v) \cdot e(u_2, v)$; and for any $u, v \in G_2$, $e(\psi(u), v) = e(\psi(v), u)$.

Definition 2.3. Two groups (G_1, G_2) are a bilinear group pair if the group action on either can be computed in one time unit, the map ψ from G_2 to G_1 can be computed in one time unit, a bilinear map $e : G_1 \times G_2 \to G_T$ exists, and e is computable in one time unit.

Definition 2.4. Two groups (G_1, G_2) are a (t, ϵ)-bilinear group pair for co-Diffie-Hellman if they are a bilinear group pair and no algorithm (t, ϵ)-breaks Computational co-Diffie-Hellman on them.

Joux and Nguyen [17] showed that an efficiently-computable bilinear map e provides an algorithm for solving the decision co-Diffie-Hellman problem. For a tuple (g_2, g_2^a, h, h^b) we have

$$a = b \bmod p \quad \iff \quad e(h, g_2^a) = e(h^b, g_2) \ .$$

Consequently, if two groups (G_1, G_2) are a (t, ϵ)-bilinear group pair for co-Diffie-Hellman, then they are also a $(t/2, \epsilon)$-co-GDH group pair. The converse is probably not true.

2.2 The Co-GDH Signature Scheme

We review the signature scheme of [6], which can be based on any gap group. It comprises three algorithms, *KeyGen*, *Sign*, and *Verify*, and uses a full-domain hash function $H : \{0,1\}^* \to G_1$, viewed as a random oracle [3].

Key Generation. Pick random $x \stackrel{\mathrm{R}}{\leftarrow} \mathbb{Z}_p$, and compute $v \leftarrow g_2^x$. The public key is $v \in G_2$. The secret key is $x \in \mathbb{Z}_p$.

Signing. Given a secret key x and a message $M \in \{0,1\}^*$, compute $h \leftarrow H(M)$, where $h \in G_1$, and $\sigma \leftarrow h^x$. The signature is $\sigma \in G_1$.

Verification. Given a public key v, a message M, and a signature σ, compute $h \leftarrow H(M)$ and verify that (g_2, v, h, σ) is a valid co-Diffie-Hellman tuple.

A co-GDH signature is a single element of G_1. On certain elliptic curves these signatures are very short: they are half the size of DSA signatures with similar security. Theorem 1 of [6] proves the existential unforgeability of the scheme under a chosen message attack [14] in the random oracle model assuming (G_1, G_2) is a co-gap group pair for Diffie-Hellman.

3 Aggregate Signatures

We define aggregate signatures and describe an aggregate signature scheme based on co-GDH signatures. Unlike the co-GDH scheme, aggregate signatures require the existence of a bilinear map. We define security models and provide proofs of security for aggregate signatures.

Consider a set $\mathbb{U}$ of users. Each user $u \in \mathbb{U}$ has a signing keypair (PK_u, SK_u). We wish to aggregate the signatures of some subset $U \subseteq \mathbb{U}$. Each user $u \in U$ produces a signature σ_u on a message M_u of her choice. These signatures are then combined into a single aggregate σ by an aggregating party. The aggregating party, who can be different from and untrusted by the users in U, has access to the users' public keys, to the messages, and to the signatures on them, but not

to any private keys. The result of this aggregation is an aggregate signature σ whose length is the same as that of any of the individual signatures. This aggregate has the property that a verifier given σ along with the identities of the parties involved and their respective messages is convinced that each user signed her respective message.

3.1 Bilinear Aggregate Signatures

We describe a bilinear aggregate signature scheme based on the co-GDH scheme presented above. Individual signatures in the aggregate signature scheme are created and verified precisely as are signatures in the co-GDH scheme (Sect. 2.2). Aggregate verification makes use of a bilinear map on G_1 and G_2.

The aggregate signature scheme allows the creation of signatures on arbitrary distinct messages $M_i \in \{0,1\}^*$. An individual signature σ_i is an element of G_1. The base groups G_1 and G_2, their respective generators g_1 and g_2, the computable isomorphism ψ from G_2 to G_1, and the bilinear map $e : G_1 \times G_2 \to G_T$, with target group G_T, are system parameters.

The scheme comprises five algorithms: *KeyGen*, *Sign*, *Verify*, *Aggregate*, and *AggregateVerify*. The first three are as in ordinary signature schemes; the last two provide the aggregation capability. The scheme employs a full-domain hash function $H : \{0,1\}^* \to G_1$, viewed as a random oracle.

Key Generation. For a particular user, pick random $x \stackrel{\mathrm{R}}{\leftarrow} \mathbb{Z}_p$, and compute $v \leftarrow g_2^x$. The user's public key is $v \in G_2$. The user's secret key is $x \in \mathbb{Z}_p$.

Signing. For a particular user, given the secret key x and a message $M \in \{0,1\}^*$, compute $h \leftarrow H(M)$, where $h \in G_1$, and $\sigma \leftarrow h^x$. The signature is $\sigma \in G_1$.

Verification. Given user's public key v, a message M, and a signature σ, compute $h \leftarrow H(M)$; accept if $e(\sigma, g_2) = e(h, v)$ holds.

Aggregation. For the aggregating subset of users $U \subseteq \mathbb{U}$, assign to each user an index i, ranging from 1 to $k = |U|$. Each user $u_i \in U$ provides a signature $\sigma_i \in G_1$ on a message $M_i \in \{0,1\}^*$ of his choice. The messages M_i must all be distinct. Compute $\sigma \leftarrow \prod_{i=1}^{k} \sigma_i$. The aggregate signature is $\sigma \in G_1$.

Aggregate Verification. We are given an aggregate signature $\sigma \in G_1$ for an aggregating subset of users U, indexed as before, and are given the original messages $M_i \in \{0,1\}^*$ and public keys $v_i \in G_2$ for all users $u_i \in U$. To verify the aggregate signature σ,

1. ensure that the messages M_i are all distinct, and reject otherwise; and
2. compute $h_i \leftarrow H(M_i)$ for $1 \le i \le k = |U|$, and accept if $e(\sigma, g_2) = \prod_{i=1}^{k} e(h_i, v_i)$ holds.

A bilinear aggregate signature, like a co-GDH signature, is a single element of G_1. Note that aggregation can be done incrementally.

The intuition behind bilinear aggregate signatures is as follows. Each user u_i has a secret key $x_i \in \mathbb{Z}_p$ and a public key $v_i = g_2^{x_i}$. User u_i's signature, if correctly formed, is $\sigma_i = h_i^{x_i}$, where h_i is the hash of the user's chosen message, M_i. The aggregate signature σ is thus $\sigma = \prod_i \sigma_i = \prod_i h_i^{x_i}$. Using the properties of the bilinear map, the left-hand side of the verification equation expands:

$$e(\sigma, g_2) = e(\prod\nolimits_i h_i^{x_i}, g_2) = \prod\nolimits_i e(h_i, g_2)^{x_i} = \prod\nolimits_i e(h_i, g_2^{x_i}) = \prod\nolimits_i e(h_i, v_i) \ ,$$

which is the right-hand side, as required. It remains to prove the security of the scheme.

3.2 Aggregate Signature Security

Informally, the security of aggregate signature schemes is equivalent to the nonexistence of an adversary capable, within the confines of a certain game, of existentially forging an aggregate signature. Existential forgery here means that the adversary attempts to forge an aggregate signature, on messages of his choice, by some set of users.

We formalize this intuition as the aggregate chosen-key security model. In this model, the adversary $\mathcal{A}$ is given a single public key. His goal is the existential forgery of an aggregate signature. We give the adversary power to choose all public keys except the challenge public key. The adversary is also given access to a signing oracle on the challenge key. His advantage, $\mathsf{Adv}\,\mathsf{AggSig}_{\mathcal{A}}$, is defined to be his probability of success in the following game.

> **Setup.** The aggregate forger $\mathcal{A}$ is provided with a public key PK_1, generated at random.
>
> **Queries.** Proceeding adaptively, $\mathcal{A}$ requests signatures with PK_1 on messages of his choice.
>
> **Response.** Finally, $\mathcal{A}$ outputs $k-1$ additional public keys $PK_2, \ldots, PK_k$. Here k is at most N, a game parameter. These keys, along with the initial key PK_1, will be included in $\mathcal{A}$'s forged aggregate. $\mathcal{A}$ also outputs messages $M_1, \ldots, M_k$; and, finally, an aggregate signature σ by the k users, each on his corresponding message.
>
> The forger wins if the aggregate signature σ is a valid aggregate on messages $M_1, \ldots, M_k$ under keys $PK_1, \ldots, PK_k$, and σ is nontrivial, i.e., $\mathcal{A}$ did not request a signature on M_1 under PK_1. The probability is over the coin tosses of the key-generation algorithm and of $\mathcal{A}$.

Definition 3.1. An aggregate forger $\mathcal{A}$ $(t, q_H, q_S, N, \epsilon)$-breaks an N-user aggregate signature scheme in the aggregate chosen-key model if: $\mathcal{A}$ runs in time at most t; $\mathcal{A}$ makes at most q_H queries to the hash function and at most q_S queries to the signing oracle; $\mathsf{Adv}\,\mathsf{AggSig}_{\mathcal{A}}$ is at least ϵ; and the forged aggregate signature is by at most N users. An aggregate signature scheme is $(t, q_H, q_S, N, \epsilon)$-secure against existential forgery in the aggregate chosen-key model if no forger $(t, q_H, q_S, N, \epsilon)$-breaks it.

A potential attack on aggregate signatures. The adversary's ability in the chosen-key model to generate keys suggests the following attack, previously considered in the context of multisignatures [20, 4]. Alice publishes her public key v_A. Bob generates a private key x'_B and a public key $v'_B = g_2^{x'_B}$, but publishes as his public key $v_B = v'_B / v_A$, a value whose discrete log he does not know. Then $H(M)^{x'_B}$ verifies as an aggregate signature on M by both Alice and Bob. Note that in this forgery Alice and Bob both sign the same message M.

One countermeasure is to require the adversary to prove knowledge of the discrete logarithms (to base g_2) of his published public keys. For example, Boldyreva, in her multisignature scheme [4], requires, in effect, that the adversary disclose the corresponding private keys $x_2, \ldots, x_k$. Micali et al. [20] discuss a series of more sophisticated approaches based on zero-knowledge proofs, again with the effect that the adversary is constrained in his key selection. These defenses apply equally well to our aggregate signature scheme. For aggregate signatures, though, there is a simpler defense.

A simple defense for aggregate signatures. In the context of aggregate signatures we can defend against the attack above by simply requiring that an aggregate signature is valid only if it is an aggregation of signatures on *distinct* messages. This restriction, codified in Step 1 of *AggregateVerify*, suffices to prove the security of the bilinear aggregate signature scheme in the chosen-key model. There is no need for zero-knowledge proofs or the disclosure of private keys.

The requirement that all messages in an aggregate be distinct is naturally satisfied for the applications to certificate chains and SBGP we have in mind. Even in more general environments it is easy to ensure that all messages are distinct: The signer simply prepends her public key to every message she signs prior to the application of the hash function H. The implicit prefix need not be transmitted with the signature, so signature and message length is unaffected.

The next theorem shows that this simple constraint is sufficient for proving security in the chosen-key model.

Theorem 3.2. *Let (G_1, G_2) be a (t', ϵ')-bilinear group pair for co-Diffie-Hellman, with each group of order p, with respective generators g_1 and g_2, with an isomorphism ψ computable from G_2 to G_1, and with a bilinear map $e : G_1 \times G_2 \to G_T$. Then the bilinear aggregate signature scheme on (G_1, G_2) is $(t, q_H, q_S, N, \epsilon)$-secure against existential forgery in the aggregate chosen-key model for all t and ϵ satisfying*

$$\epsilon \geq e(q_S + N) \cdot \epsilon' \qquad \text{and} \qquad t \leq t' - c_{G_1}(q_H + 2q_S + N + 4) - (N + 1) \ ,$$

where e is the base of natural logarithms, and exponentiation and inversion on G_1 take time c_{G_1}.

Proof. Suppose $\mathcal{A}$ is a forger algorithm that $(t, q_S, q_H, N, \epsilon)$-breaks the signature scheme. We show how to construct a t'-time algorithm $\mathcal{C}$ that solves co-CDH in (G_1, G_2) with probability at least ϵ'. This will contradict the fact that (G_1, G_2) are a (t', ϵ')-co-GDH group pair.

Let g_2 be a generator of G_2. Algorithm $\mathcal{C}$ is given $g_2, u \in G_2$ and $h \in G_1$, where $u = g_2^a$. Its goal is to output $h^a \in G_1$. Algorithm $\mathcal{C}$ simulates the challenger and interacts with forger $\mathcal{A}$ as follows.

Setup. Algorithm $\mathcal{C}$ starts by giving $\mathcal{A}$ the generator g_2 and the public key $v_1 = u \cdot g_2^r \in G_2$, where r is random in $\mathbb{Z}_p$.

Hash Queries. At any time algorithm $\mathcal{A}$ can query the random oracle H. To respond to these queries, $\mathcal{C}$ maintains a list of tuples $\langle M^{(i)}, w^{(i)}, b^{(i)}, c^{(i)} \rangle$ as explained below. We refer to this list as the H-list. The list is initially empty. When $\mathcal{A}$ queries the oracle H at a point $M \in \{0,1\}^*$, algorithm $\mathcal{C}$ responds as follows:

1. If the query M already appears on the H-list in some tuple $\langle M, w, b, c \rangle$ then algorithm $\mathcal{C}$ responds with $H(M) = w \in G_1$.
2. Otherwise, $\mathcal{C}$ generates a random coin $c \in \{0,1\}$ so that $\Pr[c = 0] = 1/(q_S + N)$.
3. Algorithm $\mathcal{C}$ picks a random $b \in \mathbb{Z}_p$. If $c = 0$ holds, $\mathcal{C}$ computes $w \leftarrow h \cdot \psi(g_2)^b \in G_1$. If $c = 1$ holds, $\mathcal{C}$ computes $w \leftarrow \psi(g_2)^b \in G_1$.
4. Algorithm $\mathcal{C}$ adds the tuple $\langle M, w, b, c \rangle$ to the H-list and responds to $\mathcal{A}$ as $H(M) = w$.

Note that, either way, w is uniform in G_1 and is independent of $\mathcal{A}$'s current view as required.

Signature queries. Algorithm $\mathcal{A}$ requests a signature on some message M under the challenge key v_1. Algorithm $\mathcal{C}$ responds to this query as follows:

1. Algorithm $\mathcal{C}$ runs the above algorithm for responding to H-queries on M, obtaining the corresponding tuple $\langle M, w, b, c \rangle$ on the H-list. If $c = 0$ holds then $\mathcal{C}$ reports failure and terminates.
2. We know that $c = 1$ holds and hence $w = \psi(g_2)^b \in G_1$. Let $\sigma = \psi(u)^b \cdot \psi(g_2)^{rb} \in G_1$. Observe that $\sigma = w^{a+r}$ and therefore σ is a valid signature on M under the public key $v_1 = u \cdot g_2^r = g_2^{a+r}$. Algorithm $\mathcal{C}$ gives σ to algorithm $\mathcal{A}$.

Output. Finally, $\mathcal{A}$ halts. It either concedes failure, in which case so does $\mathcal{C}$, or it returns a value k (where $k \le N$), $k-1$ public keys $v_2, \ldots, v_k \in G_2$, k messages $M_1, \ldots M_k$, and a forged aggregate signature $\sigma \in G_1$. The messages M_i must all be distinct, and $\mathcal{A}$ must not have requested a signature on M_1. Algorithm $\mathcal{C}$ runs its hash algorithm at each M_i, $1 \le i \le k$, obtaining the k corresponding tuples $\langle M_i, w_i, b_i, c_i \rangle$ on the H-list.

Algorithm $\mathcal{C}$ now proceeds only if $c_1 = 0$ and, for $2 \le i \le k$, $c_i = 1$; otherwise $\mathcal{C}$ declares failure and halts. Since $c_1 = 0$, it follows that $w_1 = h \cdot \psi(g_2)^{b_1}$. For $i > 1$, since $c_i = 1$, it follows that $w_i = \psi(g_2)^{b_i}$. The aggregate signature σ must satisfy the aggregate verification equation, $e(\sigma, g_2) = \prod_{i=1}^{k} e(w_i, v_i)$. For each $i > 1$, $\mathcal{C}$ sets $\sigma_i \leftarrow \psi(v_i)^{b_i}$. Then, for $i > 1$,

$$e(\sigma_i, g_2) = e(\psi(v_i)^{b_i}, g_2) = e(\psi(v_i), g_2)^{b_i} = e(\psi(g_2), v_i)^{b_i} = e(\psi(g_2)^{b_i}, v_i) = e(w_i, v_i) \ ,$$

So σ_i is a valid signature on M_i (whose hash is w_i) by the key whose public component is v_i. Now $\mathcal{C}$ constructs a value σ_1: $\sigma_1 \leftarrow \sigma \cdot \left(\prod_{i=2}^{k} \sigma_i\right)^{-1}$. Then

$$e(\sigma_1, g_2) = e(\sigma, g_2) \cdot \prod_{i=2}^{k} e(\sigma_i, g_2)^{-1} = \prod_{i=1}^{k} e(w_i, v_i) \cdot \prod_{i=2}^{k} e(w_i, v_i)^{-1} = e(w_1, v_1) \ .$$

Thus σ_1 is a valid co-GDH signature by key $v_1 = u \cdot g_2^r = g_2^{a+r}$ on a message whose hash is $w_1 = h \cdot \psi(g_2)^{b_1}$. Then $\mathcal{C}$ calculates and outputs the required h^a as $h^a \leftarrow \sigma_1 \cdot (\psi(u)^{b_1} \cdot h^r \cdot \psi(g_2)^{r b_1})^{-1}$.

This completes the description of algorithm $\mathcal{C}$. It remains to show that $\mathcal{C}$ solves the given instance of the co-CDH problem in (G_1, G_2) with probability at least ϵ'. To do so, we analyze the three events needed for $\mathcal{C}$ to succeed:

$\mathcal{E}_1$**:** $\mathcal{C}$ does not abort as a result of any of $\mathcal{A}$'s signature queries.

$\mathcal{E}_2$**:** $\mathcal{A}$ generates a valid and nontrivial aggregate signature forgery $(k, v_2, \ldots, v_k, M_1, \ldots, M_k, \sigma)$.

$\mathcal{E}_3$**:** Event $\mathcal{E}_2$ occurs, and, in addition, $c_1 = 0$, and, for $2 \le i \le k$, $c_i = 1$, where for each i c_i is the c-component of the tuple containing M_i on the H-list.

$\mathcal{C}$ succeeds if all of these events happen. The probability $\Pr[\mathcal{E}_1 \wedge \mathcal{E}_3]$ decomposes as

$$\Pr[\mathcal{E}_1 \wedge \mathcal{E}_3] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \Pr[\mathcal{E}_3 \mid \mathcal{E}_1 \wedge \mathcal{E}_2]. \tag{1}$$

The following claims give a lower bound for each of these terms.

Claim 3.3. *The probability that algorithm $\mathcal{C}$ does not abort as a result of $\mathcal{A}$'s aggregate signature queries is at least $(1 - 1/(q_S + N))^{q_S}$. Hence, $\Pr[\mathcal{E}_1] \ge (1 - 1/(q_S + N))^{q_S}$.*

Proof. Without loss of generality we assume that $\mathcal{A}$ does not ask for the signature of the same message twice. We prove by induction that after $\mathcal{A}$ makes ℓ signature queries the probability that $\mathcal{C}$ does not abort is at least $(1 - 1/(q_S + N))^{\ell}$. The claim is trivially true for $\ell = 0$. Let $M^{(\ell)}$ be $\mathcal{A}$'s ℓ'th signature query and let $\langle M^{(\ell)}, w^{(\ell)}, b^{(\ell)}, c^{(\ell)} \rangle$ be the corresponding tuple on the H-list. Then, prior to $\mathcal{A}$'s issuing the query, the bit $c^{(\ell)}$ is independent of $\mathcal{A}$'s view — the only value that could be given to $\mathcal{A}$ that depends on $c^{(\ell)}$ is $H(M^{(\ell)})$, but the distribution of $H(M^{(\ell)})$ is the same whether $c^{(\ell)} = 0$ or $c^{(\ell)} = 1$. Therefore, the probability that this query causes $\mathcal{C}$ to abort is at most $1/(q_S + N)$. Using the inductive hypothesis and the independence of $c^{(\ell)}$, the probability that $\mathcal{C}$ does not abort after this query is at least $(1 - 1/(q_S + N))^{\ell}$. This proves the inductive claim. Since $\mathcal{A}$ makes at most q_S signature queries the probability that $\mathcal{C}$ does not abort as a result of all signature queries is at least $(1 - 1/(q_S + N))^{q_S}$. □

Claim 3.4. *If algorithm $\mathcal{C}$ does not abort as a result of $\mathcal{A}$'s queries then algorithm $\mathcal{A}$'s view is identical to its view in the real attack. Hence,* $\Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \geq \epsilon$.

Proof. The public key given to $\mathcal{A}$ is from the same distribution as public keys produced by algorithm *KeyGen*. Responses to hash queries are as in the real attack since each response is uniformly and independently distributed in G_1. Since $\mathcal{C}$ did not abort as a result of $\mathcal{A}$'s signature queries, all its responses to those queries are valid. Therefore $\mathcal{A}$ will produce a valid and nontrivial aggregate signature forgery with probability at least ϵ. Hence $\Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \geq \epsilon$. □

Claim 3.5. *The probability that algorithm $\mathcal{C}$ does not abort after $\mathcal{A}$ outputs a valid and nontrivial forgery is at least* $(1 - 1/(q_S + N))^{N-1} \cdot 1/(q_S + N)$.
Hence, $\Pr[\mathcal{E}_3 \mid \mathcal{E}_1 \wedge \mathcal{E}_2] \geq (1 - 1/(q_S + N))^{N-1} \cdot 1/(q_S + N)$.

Proof. Events $\mathcal{E}_1$ and $\mathcal{E}_2$ have occurred, and $\mathcal{A}$ has generated some valid and nontrivial forgery $(k, v_2, \ldots, v_k, M_1, \ldots, M_k, \sigma)$. For each i, $1 \leq i \leq k$, let $\langle M_i, w_i, b_i, c_i \rangle$ be the tuple corresponding to M_i on the H-list. Algorithm $\mathcal{C}$ will abort unless $\mathcal{A}$ generates a forgery such that $c_1 = 0$ and, for $i > 1$, $c_i = 1$.

Since all the messages $M_1, M_2, \ldots, M_k$ are distinct, the values $c_1, c_2, \ldots, c_k$ are all independent of each other; as before, $H(M_i) = w_i$ is independent of c_i for each i.

Since its forgery is nontrivial, $\mathcal{A}$ cannot have asked for a signature on M_1 under key v_1. It can thus have no information about the value of c_1; in the forged aggregate, $c_1 = 0$ occurs with probability $1/(q_S + N)$. For each $i > 1$, $\mathcal{A}$ either asked for a signature under key v_1 on M_i, in which case $c_i = 1$ with probability 1, or it didn't, and $c_i = 1$ with probability $1 - 1/(q_S + N)$. Regardless, the probability that $c_i = 1$ for all i, $2 \leq i \leq k$, is at least $(1 - 1/(q_S + N))^{k-1} \geq (1 - 1/(q_S + N))^{N-1}$.

Therefore $\Pr[\mathcal{E}_3 \mid \mathcal{E}_1 \wedge \mathcal{E}_2] \geq (1 - 1/(q_S + N))^{N-1} \cdot 1/(q_S + N)$, as required. □

To complete the proof of Theorem 3.2, we use the bounds from the claims above in equation (1). Algorithm $\mathcal{C}$ produces the correct answer with probability at least

$$\left(1 - \frac{1}{q_S + N}\right)^{q_S + N - 1} \cdot \frac{1}{q_S + N} \cdot \epsilon \geq \frac{\epsilon / e}{q_S + N} \geq \epsilon' \ ,$$

as required.

Algorithm $\mathcal{C}$'s running time is the same as $\mathcal{A}$'s running time plus the time is takes to respond to $(q_H + q_S)$ hash queries and q_S signature queries, and the time to transform $\mathcal{A}$'s final forgery into the co-CDH solution. Each query requires an exponentiation in G_1. The output phase requires at most N additional hash computations, two inversions, two exponentiations, and $N + 1$ multiplications. We assume that exponentiation and inversion in G_1 take time c_{G_1}. Hence, the total running time is at most $t + c_{G_1}(q_H + 2q_S + N + 4) + N + 1 \leq t'$ as required. This completes the proof of Theorem 3.2. □

Aggregate verification time. Let σ be an aggregate of the n signatures $\sigma_1, \ldots, \sigma_n$. The time to verify the aggregate signature σ is linear in n. In the special case when all n signatures are issued by the same public key v, aggregate verification is faster. One need only verify that $e(\sigma, g_2) = e(v, \prod_{i=1}^{n} H(M_i))$ holds, where $M_1, \ldots, M_n$ are the signed messages.

4 Verifiably Encrypted Signatures

Next, we show an application of aggregate signatures to verifiably encrypted signatures. Verifiably encrypted signatures are used in applications such as online contract signing [1, 2]. Suppose Alice wants to show Bob that she has signed a message, but does not want Bob to possess her signature of that message. (Alice will give her signature to Bob only when a certain event has occurred, e.g., Bob has given Alice his signature on the same message.) Alice can achieve this by encrypting her signature using the public key of a trusted third party, and sending this to Bob along with a proof that she has given him a valid encryption of her signature. Bob can verify that Alice has signed the message, but cannot deduce any information about her signature. Later in the protocol, if Alice is unable or unwilling to reveal her signature, Bob can ask the third party to reveal Alice's signature. We note that the resulting contract signing protocol is not abuse-free in the sense of [10].

We show that a variant of the bilinear aggregate signature scheme allows the creation of very efficient verifiably encrypted signatures.

4.1 Verifiably Encrypted Signature Security

A verifiably encrypted signature scheme comprises seven algorithms. Three, *KeyGen*, *Sign*, and *Verify*, are analogous to those in ordinary signature schemes. The others, *AdjKeyGen*, *VESigCreate*, *VESigVerify*, and *Adjudicate*, provide the verifiably encrypted signature capability. The algorithms are described below. We refer to the trusted third party as the adjudicator.

Key Generation, Signing, Verification. As in standard signature schemes.

Adjudicator Key. Generate a public-private key pair (APK, ASK) for the adjudicator.

VESig Creation. Given a secret key SK, a message M, and an adjudicator's public key APK, compute (probabilistically) a verifiably encrypted signature ω on M.

VESig Verification. Given a public key PK, a message M, an adjudicator's public key APK, and a verifiably encrypted signature ω, verify that ω is a valid verifiably encrypted signature on M under key PK.

Adjudication. Given an adjudicator's keypair (APK, ASK), a certified public key PK, and a verifiably encrypted signature ω on some message M, extract and output σ, an ordinary signature on M under PK.

Besides the ordinary notions of signature security in the signature component, we require three security properties of verifiably encrypted signatures: validity, unforgeability, and opacity. We describe these properties in the single user setting.

Validity requires that verifiably encrypted signatures verify, and that adjudicated verifiably encrypted signatures verify as ordinary signatures, i.e., that $\mathit{VESigVerify}(M, \mathit{VESigCreate}(M))$ and $\mathit{Verify}(M, \mathit{Adjudicate}(\mathit{VESigCreate}(M))$ hold for all M and for all properly-generated keypairs and adjudicator keypairs. (The keys provided to the algorithms are here omitted for brevity.)

Unforgeability requires that it be difficult to forge a valid verifiably encrypted signature. The advantage in existentially forging a verifiably encrypted signature of an algorithm $\mathcal{F}$, given access to a verifiably-encrypted-signature creation oracle S and an adjudication oracle A, along with a

hash oracle, is

$$\mathsf{Adv}\,\mathsf{VSigF}_{\mathcal{F}} \stackrel{\text{def}}{=} \Pr\left[\begin{array}{l} \mathit{VESigVerify}(PK, APK, M, \omega) = \texttt{valid} : \\ \qquad (PK, SK) \stackrel{\text{R}}{\leftarrow} \mathit{KeyGen}, \\ \qquad (APK, ASK) \stackrel{\text{R}}{\leftarrow} \mathit{AdjKeyGen}, \\ \qquad (M, \omega) \stackrel{\text{R}}{\leftarrow} \mathcal{F}^{S,A}(PK, APK) \end{array}\right].$$

The probability is taken over the coin tosses of the key-generation algorithms, of the oracles, and of the forger. The forger is additionally constrained in that its forgery on M must be nontrivial: It must not previously have queried either oracle at M. Note that an ordinary signing oracle is not provided; it can be simulated by a call to S followed by a call to A.

Definition 4.1. A verifiably encrypted signature forger $\mathcal{F}$ $(t, q_H, q_S, q_A, \epsilon)$-forges a verifiably encrypted signature if: Algorithm $\mathcal{F}$ runs in time at most t; $\mathcal{F}$ makes at most q_H queries to the hash function, at most q_S queries to the verifiably-encrypted-signature creation oracle S, at most q_A queries to the adjudication oracle A; and $\mathsf{Adv}\,\mathsf{VSigF}_{\mathcal{F}}$ is at least ϵ. A verifiably encrypted signature scheme is $(t, q_H, q_S, q_A, \epsilon)$-secure against existential forgery if no forger $(t, q_H, q_S, q_A, \epsilon)$-breaks it.

Opacity requires that it be difficult, given a verifiably encrypted signature, to extract an ordinary signature on the same message. The advantage in extracting a verifiably encrypted signature of an algorithm $\mathcal{E}$, given access to a verifiably-encrypted-signature creation oracle S and an adjudication oracle A, along with a hash oracle, is

$$\mathsf{Adv}\,\mathsf{VSigE}_{\mathcal{E}} \stackrel{\text{def}}{=} \Pr\left[\begin{array}{l} \mathit{Verify}(PK, M, \sigma) = \texttt{valid} : \\ \qquad (PK, SK) \stackrel{\text{R}}{\leftarrow} \mathit{KeyGen}, \\ \qquad (APK, ASK) \stackrel{\text{R}}{\leftarrow} \mathit{AdjKeyGen}, \\ \qquad (M, \sigma) \stackrel{\text{R}}{\leftarrow} \mathcal{E}^{S,A}(PK, APK) \end{array}\right].$$

The probability is taken over the coin tosses of the key-generation algorithms, of the oracles, and of the forger. The extraction must be nontrivial: the adversary must not have queried the adjudication oracle A at M. (It is allowed, however, to query S at M.) Verifiably encrypted signature extraction is thus no more difficult than forgery in the underlying signature scheme.

Definition 4.2. An algorithm $\mathcal{E}$ $(t, q_H, q_S, q_A, \epsilon)$-extracts a verifiably encrypted signature if $\mathcal{E}$ runs in time at most t, makes at most q_H queries to the hash function, at most q_S queries to the verifiably-encrypted-signature creation oracle S, at most q_A queries to the adjudication oracle, and $\mathsf{Adv}\,\mathsf{VSigE}_{\mathcal{E}}$ is at least ϵ. A verifiably encrypted signature scheme is $(t, q_H, q_S, q_A, \epsilon)$-secure against extraction if no algorithm $(t, q_H, q_S, q_A, \epsilon)$-extracts it.

4.2 Aggregate Extraction

Our verifiably encrypted signature scheme depends on the assumption that given an aggregate signature of k signatures it is difficult to extract the individual signatures.

Consider the bilinear aggregate signature scheme on a group pair (G_1, G_2). We posit that it is difficult to recover the individual signatures σ_i given their aggregate σ, the public keys, and the message hashes. In fact, we posit that it is difficult to recover an aggregate σ' of any proper subset of the signatures. This we term the k-element aggregate extraction problem.

We formalize this assumption as follows. Let (G_1, G_2) be a bilinear group pair for co-Diffie-Hellman, each of order p, with respective generators g_1 and g_2, a computable isomorphism $\psi : G_2 \to G_1$ such that $g_1 = \psi(g_2)$, and a computable bilinear map $e : G_1 \times G_2 \to G_T$.

Consider a k-user aggregate in this setting. Each user has a private key $x_i \in \mathbb{Z}_p$ and a public key $v_i = g_2^{x_i} \in G_2$. Each user selects a distinct message $M_i \in \{0,1\}^*$ whose hash is $h_i \in G_1$ and creates a signature $\sigma_i = h_i^{x_i} \in G_1$. Finally, the signatures are aggregated, yielding $\sigma = \prod_i \sigma_i \in G_1$.

Let I be the set $\{1, \ldots, k\}$. Each public key v_i can be expressed as $g_2^{x_i}$, each hash h_i as $g_1^{y_i}$, each signature σ_i as $g_1^{x_i y_i}$, and the aggregate signature σ as g_1^z, where $z = \sum_{i \in I} x_i y_i$. The advantage of an algorithm $\mathcal{E}$ in extracting a subaggregate from a k-element aggregate is

$$\mathsf{Adv}\, k\text{-}\mathsf{Extr}_{\mathcal{E}} \stackrel{\text{def}}{=} \Pr\left[\begin{array}{l} (\emptyset \neq I' \subsetneq I) \wedge (\sigma' = g_1^{(\sum_{i \in I'} x_i y_i)}) : \\ \qquad x_1, \ldots, x_k, y_1, \ldots, y_k \stackrel{\text{R}}{\leftarrow} \mathbb{Z}_p,\ \sigma \leftarrow g_1^{(\sum_{i \in I} x_i y_i)}, \\ \qquad (\sigma', I') \stackrel{\text{R}}{\leftarrow} \mathcal{E}(g_2^{x_1}, \ldots, g_2^{x_k}, g_1^{y_1}, \ldots, g_1^{y_k}, \sigma) \end{array}\right].$$

The probability is taken over the choices of all x_i and y_i, and the coin tosses of $\mathcal{E}$.

Definition 4.3. An algorithm $\mathcal{E}$ (t, k, ϵ)-extracts a subaggregate from an k-element bilinear aggregate signature if $\mathcal{E}$ runs in time at most t and $\mathsf{Adv}\, k\text{-}\mathsf{Extr}_{\mathcal{E}}$ is at least ϵ. An instantiation of the bilinear aggregate signature scheme is (t, k, ϵ)-secure against aggregate extraction if no algorithm (t, k, ϵ)-extracts it.

We will be particularly concerned with the case $k = 2$. In this case, the aggregate extraction problem reduces to this one: given g_2^a, g_2^b, g_1^u, g_1^v, and g_1^{au+bv}, calculate g_1^{au}. (If the extractor outputs g_1^{bv} instead, we may recover g_1^{au} as g_1^{au+bv}/g_1^{bv}.)

4.3 Verifiably Encrypted Signatures via Aggregation

We motivate our construction for verifiably encrypted signatures by considering aggregate signatures as a launching point. An aggregate signature scheme can give rise to a verifiably encrypted signature scheme if it is difficult to extract individual signatures from an aggregate, but easy to forge existentially under the adjudicator's key. Consider the following:

1. Alice wishes to create a verifiably encrypted signature, which Bob will verify; Carol is the adjudicator. Alice and Carol's keys are both generated under the underlying signature scheme's key-generation algorithm.

2. Alice creates a signature σ on M under her public key. She forges a signature σ' on some random message M' under Carol's public key. She then combines σ and σ', obtaining an aggregate ω. The verifiably encrypted signature is the pair (ω, M').

3. Bob validates Alice's verifiably encrypted signature (ω, M') on M by checking that ω is a valid aggregate signature by Alice on M and by Carol on M'.

4. Carol adjudicates, given a verifiably encrypted signature (ω, M') on M by Alice, by computing a signature σ' on M' under her key, and removing σ' from the aggregate; what remains is Alice's ordinary signature σ.

In the bilinear aggregate signature scheme, it is difficult to extract individual signatures, under the aggregate extraction assumption. Moreover, existential forgery is easy when the random oracle hash function is set aside: Given a public key $v \in G_2$ and $r \in \mathbb{Z}_p$, $\psi(v)^r$ is a valid signature on a message whose hash is $\psi(g_2)^r = g_1^r$. Below, we formalize and prove the security of the verifiably encrypted signature scheme created in this way.

4.4 The Bilinear Verifiably-Encrypted Signature Scheme

The bilinear verifiably encrypted signature scheme is built on the bilinear aggregate signature scheme of the previous section. It shares the key-generation algorithm with the underlying aggregate scheme. Moreover, the adjudicator's public and private information is simply an aggregate-signature keypair. The scheme comprises the seven algorithms described below:

Key Generation. *KeyGen* and *AdjKeyGen* are the same as *KeyGen* in the co-GDH signature scheme.

Signing, Verification. *Sign* and *Verify* are the same as in the co-GDH signature scheme.

VESig Creation. Given a secret key $x \in \mathbb{Z}_p$ a message $M \in \{0,1\}^*$, and an adjudicator's public key $v' \in G_2$, compute $h \leftarrow H(M)$, where $h \in G_1$, and $\sigma \leftarrow h^x$. Select r at random from $\mathbb{Z}_p$ and set $\mu \leftarrow \psi(g_2)^r$ and $\sigma' \leftarrow \psi(v')^r$. Aggregate σ and σ' as $\omega \leftarrow \sigma\sigma' \in G_1$. The verifiably encrypted signature is the pair (ω, μ). (This can also be viewed as ElGamal encryption of σ under the adjudicator's key.)

VESig Verification. Given a public key v, a message M, an adjudicator's public key v', and a verifiably encrypted signature (ω, μ), set $h \leftarrow H(M)$; accept if $e(\omega, g_2) = e(h, v) \cdot e(\mu, v')$ holds.

Adjudication. Given an adjudicator's public key v' and corresponding private key $x' \in \mathbb{Z}_p$, a certified public key v, and a verifiably encrypted signature (ω, μ) on some message M, ensure that the verifiably encrypted signature is valid; then output $\sigma = \omega/\mu^{x'}$.

If the adjudicator does not first validate a purported verifiably encrypted signature, a malicious user can trick him into signing arbitrary messages under his adjudication key. Similarly, the adjudicator should only adjudicate for certified public keys v; we assume that the CA, in issuing a certificate on v, verifies that the user knows the private key for v.

It is easy to see that validity holds. A verifiably encrypted signature correctly validates under *VESigVerify*, which is simply the aggregate signature verification algorithm. Moreover, for any valid verifiably encrypted signature, $e(\omega/\mu^{x'}, g_2) = e(\omega, g_2) \cdot e(\mu, g_2)^{-x'} = e(h, v) \cdot e(\mu, v') \cdot e(\mu, v')^{-1} = e(h, v)$, so the output of *Adjudicate* is a valid signature on message M under the key v.

The next two theorems prove the unforgeability and opacity of the scheme.

Theorem 4.4. *Let G_1 and G_2 be cyclic groups of prime order p, with respective generators g_1 and g_2, with a computable bilinear map $e : G_1 \times G_2 \to G_T$. Suppose that the co-GDH signature scheme is $(t', q'_H, q'_S, \epsilon')$-secure against existential forgery on (G_1, G_2). Then the bilinear verifiably encrypted signature scheme is $(t, q_H, q_S, q_A, \epsilon)$-secure against existential forgery on (G_1, G_2), for all $q_H \leq q'_H$, $q_S \leq q'_S$, $\epsilon \geq \epsilon'$, and all t satisfying $t \leq t' - 2c_{G_1}(q_S + q_A + 1)$, where exponentiation and inversion on G_1 take time c_{G_1}.*

Proof. Given a verifiably-encrypted-signature forger algorithm $\mathcal{V}$, we construct a forger algorithm $\mathcal{F}$ for the underlying co-GDH signature scheme.

We assume that $\mathcal{V}$ is well-behaved in the sense that it always requests the hash of a message M before it requests a verifiably encrypted signature or an adjudication involving M, and that it never requests adjudication on a message M on which it had not previously asked for a verifiably encrypted signature. It is trivial to modify any forger algorithm $\mathcal{V}$ to have the first property. The second property is reasonable since the input to the adjudication oracle in this case would be a nontrivial verifiably encrypted signature forgery; $\mathcal{V}$ can be modified simply to output it and halt.

The co-GDH forger $\mathcal{F}$ is given a public key v, and has access to a signing oracle for v and a hash oracle. It simulates the challenger and runs interacts with $\mathcal{V}$ as follows.

Setup. Algorithm $\mathcal{F}$ generates a key, $(x', v') \stackrel{\mathrm{R}}{\leftarrow} \mathit{KeyGen}$, which serves as the adjudicator's key. Now $\mathcal{F}$ runs $\mathcal{V}$, providing as input the public keys v and v'.

Hash Queries. Algorithm $\mathcal{V}$ requests a hash on some string M. Algorithm $\mathcal{F}$ makes a query on M to its own hash oracle, receiving some value $h \in G_1$, with which it responds to $\mathcal{V}$'s query.

VerSig Creation Queries. Algorithm $\mathcal{V}$ requests a signature on some string M. (It will have already queried the hash oracle at M.) $\mathcal{F}$ queries its signing oracle (for v) at M, obtaining $\sigma \in G_1$. It then selects r at random from $\mathbb{Z}_p$, and returns to $\mathcal{V}$ the pair $(\sigma \cdot \psi(v')^r, \psi(g_2)^r)$.

Adjudication Queries. Algorithm $\mathcal{V}$ requests adjudication for (ω, μ), a verifiably encrypted signature on a message M under key v and adjudicator key v'. Algorithm $\mathcal{F}$ checks that the verifiably encrypted signature is valid, then returns $\omega/\mu^{x'}$.

Output. Finally, $\mathcal{V}$ halts, either declaring failure, in which case $\mathcal{F}$, too, declares failure and halts, or providing a valid and nontrivial verifiably encrypted signature (ω^*, μ^*) on a message M^*. $\mathcal{F}$ sets $\sigma^* \leftarrow \omega^*/(\mu^*)^{x'}$ which, by the validity property, is a valid co-GDH signature on M^* under key v.

That the forgery is nontrivial means that $\mathcal{V}$ did not query the verifiably encrypted signature oracle at M^*, from which it follows that $\mathcal{F}$ did not query its signing oracle at M^*. Thus (M^*, σ^*) is a nontrivial co-GDH forgery; algorithm $\mathcal{F}$ outputs it and halts.

It remains only to analyze the success probability and running time of $\mathcal{F}$. Algorithm $\mathcal{F}$ succeeds whenever $\mathcal{V}$ does, that is, with probability at least ϵ.

Algorithm $\mathcal{F}$'s running time is the same as $\mathcal{V}$'s running time plus the time it takes to respond to q_H hash queries, q_S verifiably-encrypted signature queries, and q_A adjudication queries, and the time to transform $\mathcal{V}$'s final verifiably-encrypted signature forgery into a co-GDH signature forgery. Hash queries impose no overhead. Each verifiably-encrypted signature query requires $\mathcal{F}$ to perform two exponentiations in G_1. Each adjudication query requires $\mathcal{F}$ to perform an exponentiation and an inversion in G_1. The output phase also requires an exponentiation and an inversion. We assume that exponentiation and inversion in G_1 take time c_{G_1}. Hence, the total running time is at most $t + 2c_{G_1}(q_S + q_A + 1)$.

$\mathcal{F}$ queries its hash oracle whenever $\mathcal{V}$ queries its hash oracle, and its signing oracle whenever $\mathcal{V}$ queries its verifiably encrypted signature oracle.

Combining all this, we see that if $\mathcal{V}$ $(t, q_H, q_S, q_A, \epsilon)$-forges a bilinear verifiably encrypted signature on (G_1, G_2), then $\mathcal{F}$ $(t + 2c_{G_1}(q_S + q_A + 1), q_H, q_S, \epsilon)$-breaks the co-GDH signature scheme on (G_1, G_2). Conversely, if the co-GDH signature scheme is $(t', q'_H, q'_S, \epsilon')$-secure, then the bilinear verifiably encrypted signature scheme is $(t' - 2c_{G_1}(q_S + q_A + 1), q'_H, q'_S, q_A, \epsilon')$-secure against existential forgery. □

Theorem 4.5. *Let G_1 and G_2 be cyclic groups of prime order p, with respective generators g_1 and g_2, with a computable isomorphism $\psi : G_2 \to G_1$ such that $\psi(g_2) = g_1$ and a computable bilinear map $e : G_1 \times G_2 \to G_T$. Suppose that the bilinear aggregate signature scheme on (G_1, G_2) is $(t', 2, \epsilon')$-secure against aggregate extraction. Then the bilinear verifiably encrypted signature scheme is $(t, q_H, q_S, q_A, \epsilon)$-secure against extraction on (G_1, G_2) for all t and ϵ satisfying*

$$\epsilon \geq e(q_A + 1) \cdot \epsilon' \qquad \textit{and} \qquad t \leq t' - c_{G_1}(q_H + 4q_S + 2q_A + 3) \ ,$$

where e is the base of natural logarithms, and exponentiation and inversion on G_1 take time c_{G_1}.

Proof. Given a verifiably-encrypted-signature extractor algorithm $\mathcal{V}$, we construct an aggregate extractor algorithm $\mathcal{A}$. The co-GDH forger $\mathcal{A}$ is given values g_2^α and g_2^β in G_2, g_1^γ, g_1^δ, and $g_1^{\alpha\gamma+\beta\delta}$ in G_1. It runs $\mathcal{V}$, answering its oracle calls, and uses $\mathcal{V}$'s verifiably encrypted signature extraction to calculate $g_1^{\alpha\gamma}$, the answer to its own extraction challenge.

Let g_1 be a generator of G_1, and g_2 of G_2, such that $\psi(g_2) = g_1$. Algorithm $\mathcal{A}$ is given $g_2^\alpha, g_2^\beta \in G_2$ and $g_1^\gamma, g_1^\delta, g_1^{\alpha\gamma+\beta\delta} \in G_1$. Its goal is to output $g_1^{\alpha\gamma} \in G_1$. Algorithm $\mathcal{A}$ simulates the challenger and interacts with verifiably-encrypted-signature extractor $\mathcal{V}$ as follows.

Setup. Algorithm $\mathcal{A}$ sets $v \leftarrow g_2^\alpha$, the signer's public key, and $v' \leftarrow g_2^\beta$, the adjudicator's public key. It gives v and v' to $\mathcal{V}$.

Hash Queries. At any time algorithm $\mathcal{V}$ can query the random oracle H. To respond to these queries, $\mathcal{A}$ maintains a list of tuples $\langle M^{(i)}, w^{(i)}, b^{(i)}, c^{(i)} \rangle$ as explained below. We refer to this list as the H-list. The list is initially empty. When $\mathcal{V}$ queries the oracle H at a point $M \in \{0,1\}^*$, algorithm $\mathcal{A}$ responds as follows:

1. If the query M already appears on the H-list in some tuple $\langle M, w, b, c \rangle$ then algorithm $\mathcal{A}$ responds with $H(M) = w \in G_1$.
2. Otherwise, $\mathcal{A}$ generates a random coin $c \in \{0,1\}$ so that $\Pr[c = 0] = 1/(q_A + 1)$.
3. Algorithm $\mathcal{A}$ picks a random $b \in \mathbb{Z}_p$. If $c = 0$ holds, $\mathcal{A}$ computes $w \leftarrow g_1^\gamma \cdot g_1^b \in G_1$. If $c = 1$ holds, $\mathcal{A}$ computes $w \leftarrow g_1^b \in G_1$.
4. Algorithm $\mathcal{A}$ adds the tuple $\langle M, w, b, c \rangle$ to the H-list and responds to $\mathcal{V}$ as $H(M) = w$.

VerSig Creation Queries. $\mathcal{V}$ requests a verifiably-encrypted signature on some string M under challenge key v and adjudicator key v'. Algorithm $\mathcal{A}$ responds to this query as follows:

1. Algorithm $\mathcal{A}$ runs the above algorithm for responding to H-queries on M, obtaining the corresponding tuple $\langle M, w, b, c \rangle$ on the H-list.
2. $\mathcal{A}$ selects x at random from $\mathbb{Z}_p$. If c equals 0, $\mathcal{A}$ computes and returns $(\omega, \mu) = (\psi(g_2^\alpha)^b \cdot g_1^{\alpha\gamma+\beta\delta} \cdot \psi(g_2^\beta)^x, g_1^\delta \cdot g_1^x)$. If c equals 1, $\mathcal{A}$ computes and returns $(\omega, \mu) = (\psi(g_2^\alpha)^b \cdot \psi(g_2^\beta)^x, g_2^x)$. It is easy to verify that (ϵ, μ) is in either case a correct verifiably encrypted signature on the message with hash w.

Adjudication Queries. Algorithm $\mathcal{V}$ requests adjudication for (ω, μ), a verifiably encrypted signature on a message M under key v and adjudicator key v'. Algorithm $\mathcal{A}$ responds to this query as follows:

1. Algorithm $\mathcal{A}$ runs the above algorithm for responding to H-queries on M, obtaining the corresponding tuple $\langle M, w, b, c \rangle$ on the H-list.
2. Algorithm $\mathcal{A}$ checks that the verifiably encrypted signature is valid. If it is not, $\mathcal{A}$ returns $\star$, a placeholder value.
3. If c equals 0, $\mathcal{A}$ declares failure and halts. Otherwise, it computes and returns $\sigma \leftarrow \psi(g_2^\alpha)^b$. It is easy to verify that σ is the correct co-GDH signature under key v on the message with hash w.

Output. Finally, $\mathcal{V}$ halts. It either concedes failure, in which case so does $\mathcal{A}$, or returns a nontrivial extracted signature σ^* on some message M^*. For the extraction to be nontrivial, $\mathcal{V}$ must not have asked for adjudication on a verifiably encrypted signature of M^*. Algorithm $\mathcal{A}$ runs its hash algorithm at M^*, obtaining the k corresponding tuples $\langle M^*, w^*, b^*, c^* \rangle$ on the H-list.

$\mathcal{A}$ now proceeds only if $c^* = 0$; otherwise it declares failure and halts. Since $c^* = 0$, it follows that $w^* = g_1^{\gamma} \cdot g_1^{b^*}$. The extracted signature σ^* must satisfy the co-GDH verification equation, $e(\sigma^*, g_2) = e(h^*, v)$. $\mathcal{A}$ sets $\sigma \leftarrow \sigma^* / \psi(v)^{b^*}$. Then

$$\begin{aligned} e(\sigma, g_2) &= e(\sigma^*, g_2) \cdot e(\psi(v), g_2)^{-b^*} = e(w^*, v) \cdot e(\psi(g_2), v)^{-b^*} \\ &= e(g_1^{\gamma}, v) \cdot e(g_1, v)^{b^*} \cdot e(g_1, v)^{-b^*} = e(g_1^{\gamma}, g_2^{\alpha}). \end{aligned}$$

Where in the last equality we substitute $v = g_2^{\alpha}$. Thus $(g_2, g_2^{\alpha}, g_1^{\gamma}, \sigma)$ is a valid co-Diffie-Hellman tuple, so σ equals $g_2^{\alpha\gamma}$, the answer to the aggregate extraction problem; algorithm $\mathcal{A}$ outputs it and halts.

This completes the description of algorithm $\mathcal{A}$. It remains to show that $\mathcal{A}$ solves the given instance of the aggregate extraction problem on (G_1, G_2) with probability at least ϵ'. To do so, we analyze the three events needed for $\mathcal{A}$ to succeed:

$\mathcal{E}_1$**:** $\mathcal{A}$ does not abort as a result of any of $\mathcal{V}$'s adjudication queries.

$\mathcal{E}_2$**:** $\mathcal{V}$ generates a valid and nontrivial verifiably-encrypted signature extraction (M^*, σ^*).

$\mathcal{E}_3$**:** Event $\mathcal{E}_2$ occurs, and $c^* = 0$ holds, where c^* is the c-component of the tuple containing M^* on the H-list.

$\mathcal{A}$ succeeds if all of these events happen. The probability $\Pr[\mathcal{E}_1 \wedge \mathcal{E}_3]$ decomposes as

$$\Pr[\mathcal{E}_1 \wedge \mathcal{E}_3] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \Pr[\mathcal{E}_3 \mid \mathcal{E}_1 \wedge \mathcal{E}_2]. \tag{2}$$

The following claims give a lower bound for each of these terms.

Claim 4.6. *The probability that algorithm $\mathcal{A}$ does not abort as a result of $\mathcal{V}$'s adjudication queries is at least $1/e$. Hence,* $\Pr[\mathcal{E}_1] \geq 1/e$.

Proof. Without loss of generality we assume that $\mathcal{V}$ does not ask for adjudication of the same message twice. We prove by induction that after $\mathcal{V}$ makes ℓ signature queries the probability that $\mathcal{A}$ does not abort is at least $(1 - 1/(q_A + 1))^{\ell}$. The claim is trivially true for $\ell = 0$. Let $\mathcal{V}$'s ℓ'th adjudication query be for verifiably encrypted signature $(\omega^{(\ell)}, \mu^{(\ell)})$, on message $M^{(\ell)}$ under the challenge key v, and let $\langle M^{(\ell)}, w^{(\ell)}, b^{(\ell)}, c^{(\ell)} \rangle$ be the corresponding tuple on the H-list. Then prior to issuing the query, the bit $c^{(\ell)}$ is independent of $\mathcal{V}$'s view — the only values that could be given to $\mathcal{V}$ that depend on $c^{(\ell)}$ are $H(M^{(\ell)})$ and verifiably-encrypted signatures on $M^{(\ell)}$, but the distributions on these values are the same whether $c^{(\ell)} = 0$ or $c^{(\ell)} = 1$. Therefore, the probability that this query causes $\mathcal{A}$ to abort is at most $1/(q_A + 1)$. Using the inductive hypothesis and the independence of $c^{(\ell)}$, the probability that $\mathcal{A}$ does not abort after this query is at least $(1 - 1/(q_A + 1))^{\ell}$. This proves the inductive claim. Since $\mathcal{V}$ makes at most q_A adjudication queries the probability that $\mathcal{A}$ does not abort as a result of all signature queries is at least $(1 - 1/(q_A + 1))^{q_A} \geq 1/e$. □

Claim 4.7. *If algorithm $\mathcal{A}$ does not abort as a result of $\mathcal{V}$'s adjudication queries then $\mathcal{V}$'s view is identical to its view in the real attack. Hence,* $\Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \geq \epsilon$.

Proof. The challenge public key v given to $\mathcal{V}$ is from the same distribution as public keys produced by *KeyGen*; the adjudicator's public key v' given to $\mathcal{V}$ is from the same distribution as the adjudicator keys produces by *AdjKeyGen*. Responses to hash queries are as in the real attack since each response is uniformly and independently distributed in G_1. Responses to verifiably-encrypted signature queries are also as in the real attack: They are valid, and their μ components are uniformly and independently distributed in G_1. Since $\mathcal{A}$ did not abort as a result of $\mathcal{V}$'s adjudication queries, all its responses to those queries are valid. Therefore $\mathcal{V}$ will produce a valid and nontrivial verifiably-encrypted signature extraction with probability at least ϵ. Hence $\Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \geq \epsilon$. □

Claim 4.8. *The probability that algorithm $\mathcal{A}$ does not abort after $\mathcal{V}$ outputs a valid and nontrivial verifiably-encrypted signature extraction is at least $1/(q_A + 1)$ Hence,* $\Pr[\mathcal{E}_3 \mid \mathcal{E}_1 \wedge \mathcal{E}_2] \geq 1/(q_A + 1)$.

Proof. Given that events $\mathcal{E}_1$ and $\mathcal{E}_2$ happened, algorithm $\mathcal{A}$ will abort only if $\mathcal{V}$ generates a forgery (M^*, σ^*) for which the tuple $\langle M^*, w^*, b^*, c^* \rangle$ on the H-list has $c = 1$. Since its extraction is nontrivial, $\mathcal{V}$ could not have requested adjudication on any verifiably encrypted signature on M^*, and c^* must be independent of $\mathcal{V}$'s current view. Therefore $\Pr[c^* = 0 \mid \mathcal{E}_1 \wedge \mathcal{E}_2] \geq 1/(q_A + 1)$ as required. □

Using the bounds from the claims above in equation (2) shows that $\mathcal{A}$ produces the correct answer with probability at least $\epsilon/e(q_A + 1) \geq \epsilon'$ as required.

Algorithm $\mathcal{A}$'s running time is the same as $\mathcal{V}$'s running time plus the time is takes to respond to $\mathcal{A}$'s oracle queries and to transform $\mathcal{V}$'s verifiably-encrypted signature extraction into an aggregate extraction. Each verifiably-encrypted signature query, each adjudication query, and the output phase requires $\mathcal{A}$ to run its H-algorithm. It must therefore run this algorithm $(q_H + q_S + q_A + 1)$ times. Each run requires an exponentiation in G_1. Algorithm $\mathcal{A}$ must run its verifiably-encrypted signing algorithm q_S times, and each run requires at most three exponentiation in G_1. Finally, $\mathcal{A}$'s output phase requires at most one exponentiation and one inversion in G_1. We assume that exponentiation and inversion in G_1 take time c_{G_1}. Hence, the total running time is at most $t + c_{G_1}(q_H + 4q_S + 2q_A + 3) \leq t'$ as required. □

4.5 Observations on Verifiably Encrypted Signatures

We note some extensions of the verifiably encrypted signature scheme discussed above. Some of these rely for security on the k-element aggregate extraction assumption with $k > 2$.

- Anyone can convert an ordinary unencrypted signature to a verifiably encrypted signature. The same applies to unencrypted aggregate signatures.
- An adjudicator's private key can be shared amongst n parties using k-of-n threshold cryptography [12, 11], so that k parties are needed to adjudicate a verifiably encrypted signature.
- A message-signature pair in the co-GDH signature scheme is of the same form as an identity–private-key pair in the Boneh-Franklin Identity-Based Encryption Scheme [5]. Thus the verifiably encrypted signature scheme can potentially be modified to yield a verifiably encrypted encryption scheme for IBE private keys. Verifiably encrypted private keys have many applications [27].

5 Ring Signatures

Rivest, Shamir and Tauman define ring signature schemes and construct some using RSA and Rabin cryptosystems [28]. Naor defines the closely-related notion of deniable ring authentication and proposes such a scheme that relies only on the existence of a strong encryption function [23]. We shall see that co-GDH signatures give rise to natural ring signatures.

5.1 Ring Signatures

Consider a set U of users. Each user $u \in U$ has a signing keypair (PK_u, SK_u). A ring signature on U is a signature that is constructed using all the public keys of the users in U, and a single private key of any user in U. A ring signature has the property that a verifier is convinced that the signature was produced using one of the private keys of U, but is not able to determine which one. This property is called *signer-ambiguity* [28]. Applications for ring signatures include authenticated (yet repudiable) communication and leaking secrets [28].

Zhang and Kim [29] devised a bilinear ring signature in an identity-based setting. Our scheme differs from theirs, as our goal is to extend co-GDH signatures to obtain efficient ring signatures; the system parameters and key generation algorithm in our system are identical to those of the co-GDH scheme.

5.2 Bilinear Ring Signatures

The ring signature scheme comprises three algorithms: *KeyGen*, *RingSign*, and *RingVerify*. Recall g_1, g_2 are generators of groups G_1, G_2 respectively, and $e : G_1 \times G_2 \to G_T$ is a bilinear map, and a computable isomorphism $\psi : G_2 \to G_1$ exists, with $\psi(g_2) = g_1$. Again we use a full-domain hash function $H : \{0,1\}^* \to G_1$. The security analysis views H as a random oracle.

Key Generation. For a particular user, pick random $x \xleftarrow{R} \mathbb{Z}_p$, and compute $v \leftarrow g_2^x$. The user's public key is $v \in G_2$. The user's secret key is $x \in \mathbb{Z}_p$.

Ring Signing. Given public keys $v_1, \ldots, v_n \in G_2$, a message $M \in \{0,1\}^*$, and a private key x corresponding to one of the public keys v_s for some s, choose random $a_i \xleftarrow{R} \mathbb{Z}_p$ for all $i \neq s$. Compute $h \leftarrow H(M) \in G_1$ and set

$$\sigma_s \leftarrow \left(h / \psi\Big(\prod_{i \neq s} v_i^{a_i} \Big) \right)^{1/x}.$$

For all $i \neq s$ let $\sigma_i \leftarrow g_1^{a_i}$. Output the ring signature $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle \in G_1^n$.

Ring Verification. Given public keys $v_1, \ldots, v_n \in G_2$, a message $M \in \{0,1\}^*$, and a ring signature σ, compute $h \leftarrow H(M)$ and verify that $e(h, g_2) = \prod_{i=1}^n e(\sigma_i, v_i)$.

Using the bilinearity and nondegeneracy of the pairing e, it is easy to show that a signature produced by the *RingSign* algorithm will verify under the *RingVerify* algorithm.

5.3 Security

There are two aspects a security analysis for ring signatures we must consider. Firstly, signer ambiguity must be ensured. We show that the identity of the signer is unconditionally protected.

Theorem 5.1. *For any algorithm $\mathcal{A}$, any set of users U, and a random $u \in U$, the probability* $\Pr[\mathcal{A}(\sigma) = u]$ *is at most* $1/|U|$*, where σ is any ring signature on U generated with private key SK_u.*

Proof. The theorem follows from a simple probability argument: for any $h \in G_1$, and any s, $1 \leq s \leq n$, the distribution $\{g_1^{a_1}, \ldots, g_1^{a_n} : a_i \stackrel{\mathrm{R}}{\leftarrow} \mathbb{Z}_p$ for $i \neq s, a_s$ chosen such that $\prod_{i=1}^n g_1^{a_i} = h\}$ is identical to the distribution $\{g_1^{a_1}, \ldots, g_1^{a_n} : \prod_{i=1}^n g_1^{a_i} = h\}$, since the value of any one of the a_i's is uniquely determined by the values of the other a_i's. □

Secondly, we need to examine the scheme's resistance to forgery. We adopt the security model of Rivest, Shamir and Tauman [28]. Consider the following game played between an adversary and a challenger. The adversary is given the public keys $v_1, \ldots, v_n$ of a set of users U, and is given oracle access to h and a ring-signing oracle. The adversary may work adaptively. The goal of the adversary is to output a valid ring signature on U of a message M subject to the condition that M has never been presented to the ring-signing oracle. An adversary A's advantage $\mathsf{Adv}\,\mathsf{RingSig}_{\mathcal{A}}$ in existentially forging a bilinear ring signature is the probability, taken over the coin tosses of the key-generation algorithm and of the forger, that A succeeds in creating a valid ring signature in the above game.

Theorem 5.2. *Suppose $\mathcal{F}$ is a (t', ϵ')-algorithm that can produce a forgery of a ring signature on a set of users of size n. Then there exists an (t, ϵ)-algorithm that can solve the co-CDH problem where $t \leq 2t' + 2c_{G_2}(2n + q_H + nq_S)$ and $\epsilon \geq ((\epsilon'/e)(1 + q_S))^2$, where $\mathcal{F}$ issues at most q_S ring-signature queries and at most q_H hash queries, and exponentiation and inversion on G_2 take time c_{G_2}.*

Proof. The co-CDH problem can be solved by first solving two random instances of the following problem: Given g_1^{ab}, g_2^a (and g_1, g_2), compute g_1^b. We shall construct an algorithm $\mathcal{A}$ that solves this problem. This is easy if $a = 0$. In what follows, we assume $a \neq 0$.

Initially $\mathcal{A}$ picks $x_2, \ldots, x_n$ at random from $\mathbb{Z}_p$ and sets $x_1 = 1$. It sets $v_i = (g_2^a)^{x_i}$. Algorithm $\mathcal{F}$ is given the public keys $v_1, \ldots, v_n$. Without loss of generality we may assume $\mathcal{F}$ submits distinct queries (as previous replies can be cached); that for every ring-signing query on a message M, $\mathcal{F}$ has previously issued a hash query for M; and that $\mathcal{F}$ issues a hash query on the message on which it attempts to forge a signature some time before giving its final output.

On a hash query, $\mathcal{A}$ flips a coin that shows 0 with probability p and 1 otherwise (p shall be determined later). Then $\mathcal{A}$ picks a random $r \stackrel{\mathrm{R}}{\leftarrow} \mathbb{Z}_p$, and if the coins shows 0, $\mathcal{A}$ returns $(g_1^{ab})^r$, otherwise it returns $\psi(g_2^a)^r$.

Suppose $\mathcal{F}$ issues a ring sign query for a message M. By assumption, $\mathcal{A}$ has previously issued a hash query for M. If the coin $\mathcal{A}$ flipped for this h-query showed 0, then $\mathcal{A}$ fails and exits. Otherwise $\mathcal{A}$ had returned $H(M) = \psi(g_2^a)^r$ for some r. In this case $\mathcal{A}$ chooses random $a_2, \ldots, a_n \stackrel{\mathrm{R}}{\leftarrow} \mathbb{Z}_p$, computes $a_1 = r - (a_2 x_2 + \ldots + a_n x_n)$, and returns the signature $\sigma = \langle g_1^{a_1}, \ldots, g_1^{a_n} \rangle$.

Eventually $\mathcal{F}$ outputs a forgery $\langle \sigma_1, \ldots, \sigma_n \rangle$ for a message M. Again by assumption, $\mathcal{F}$ has previously issued a h-query for M. If the coin flipped by $\mathcal{A}$ for this query did not show 0 then $\mathcal{A}$ fails. Otherwise $H(M) = g_1^{abr}$ for some r chosen by $\mathcal{A}$, and $\mathcal{A}$ outputs the rth root of $\sigma_1 {\sigma_2}^{x_2} \ldots {\sigma_n}^{x_n}$.

Algorithm $\mathcal{F}$ cannot distinguish between $\mathcal{A}$'s simulation and real life. Also, $\mathcal{A}$ will not fail with probability $p^{q_S}(1 - p)$ which is maximized when $p = q_S/(q_S + 1)$, giving a bound of $(1/e)(1 + q_S)$. If it does not fail and $\mathcal{F}$ successfully forges a ring signature then $\mathcal{A}$ is successful and outputs g_1^b. Algorithm $\mathcal{A}$ requires n exponentiations on G_2 in setup, one exponentiation for each of $\mathcal{F}$'s hash queries, n exponentiations for each of $\mathcal{F}$'s signature queries, and n exponentiations in the output phase, so its running time is $\mathcal{F}$'s running time plus $c_{G_2}(2n + q_H + nq_S)$. □

5.4 Observations on Ring Signatures

Any ring signature scheme restricts to an ordinary signature scheme when $n = 1$. Our scheme restricts to a short signature scheme similar to the co-GDH scheme [6]. In this modified co-GDH scheme, σ equals $h^{1/x}$ rather than h^x, and one verifies that $e(h, g_2) = e(\sigma, v)$ rather than that $e(\sigma, g_2) = e(h, v)$.

Bresson et al. [7] extend Rivest-Shamir-Tauman ring signatures to obtain threshold and ad-hoc ring signatures. However, bilinear ring signatures have interesting properties that do not appear to be shared by ring signatures in general. For any set of users U with $u \in U$, *anyone* can convert a modified co-GDH signature by u into a ring signature by U. Specifically, to convert a modified co-GDH signature σ_1 on M for public key v_1 into a ring signature $\sigma = \langle \sigma'_1, \ldots, \sigma'_n \rangle$ on M for public keys $v_1, \ldots, v_n$, we choose $r_i \stackrel{R}{\leftarrow} \mathbb{Z}_p$ for $2 \leq i \leq n$, and set $\sigma'_1 \leftarrow \sigma_1 \prod_{i=2}^{n} \psi(v_i^{r_i})$ and $\sigma'_i \leftarrow \psi(v_1^{-r_i})$ for $2 \leq i \leq n$. More generally, anyone can further anonymize a ring signature by adding users to U.

6 Conclusions

We introduced the concept of aggregate signatures and constructed an efficient aggregate signature scheme based on bilinear maps. Key generation, aggregation, and verification require no interaction. We proved security of the system in a model that gives the adversary his choice of public keys and messages to forge. For security, we introduced the additional constraint that an aggregate signature is valid only if it is an aggregation of signatures on *distinct* messages. This constraint is satisfied naturally for the applications we have in mind. More generally, the constraint can be satisfied by prepending the public key to the message prior to signing.

We gave several applications for aggregate signatures. For example, they can be used to reduce the size of certificate chains and reduce communication bandwidth in protocols such as SBGP. We also showed that our specific aggregate signature scheme gives verifiably encrypted signatures.

Previous signature constructions using bilinear maps [6, 19, 8, 4] only required a gap Diffie-Hellman group (i.e., DDH easy, but CDH hard). The signature constructions in this paper require the extra structure provided by the bilinear map. These constructions are an example where a bilinear map provides more power than a generic gap Diffie-Hellman group.

Acknowledgments

The authors thank Leonid Reyzin, Liqun Chen, Alice Silverberg, and Cynthia Dwork for helpful discussions about this work. The first author is supported by DARPA, the Packard foundation, and an NSF CAREER award. The third and fourth authors are supported by DARPA and NSF.

References

[1] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. *IEEE J. Selected Areas in Comm.*, 18(4):593–610, April 2000.

[2] F. Bao, R. Deng, and W. Mao. Efficient and practical fair exchange protocols with offline TTP. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 77–85, 1998.

[3] M. Bellare and P. Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In *Proceedings of Eurocrypt '96*, volume 1070 of *LNCS*, pages 399–416. Springer-Verlag, 1996.

[4] A. Boldyreva. Efficient threshold signature, multisignature and blind signature schemes based on the gap-Diffie-Hellman-group signature scheme. In *Proceedings of PKC 2003*, volume 2567 of *LNCS*, pages 31–46. Springer-Verlag, 2003.

[5] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. *SIAM J. Computing*, 32(3):586–615, 2003. Extended abstract in *Proceedings of Crypto 2001.*

[6] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In *Proceedings of Asiacrypt 2001*, volume 2248 of *LNCS*, pages 514–32. Springer-Verlag, 2001. Full paper: http://crypto.stanford.edu/~dabo/pubs.html.

[7] E. Bresson, J. Stern, and M. Szydlo. Threshold ring signatures and applications to ad-hoc groups. In M. Yung, editor, *Proceedings of Crypto 2002*, volume 2442 of *LNCS*, pages 465–80. Springer-Verlag, 2002.

[8] Y. Dodis. Efficient construction of (distributed) verifiable random functions. In *Proceedings of PKC 2003*, volume 2567 of *LNCS*, pages 1–17. Springer-Verlag, 2003.

[9] A. Fiat. Batch RSA. In *Proceedings of Crypto '89*, pages 175–185, 1989.

[10] J. Garay, M. Jakobsson, and P. MacKenzie. Abuse-free optimistic contract signing. In *Proceedings of Crypto '99*, volume 1666 of *LNCS*, pages 449–466. Springer-Verlag, 1999.

[11] P. Gemmel. An introduction to threshold cryptography. *RSA CryptoBytes*, 2(3):7–12, 1997.

[12] R. Gennaro, T. Rabin, S. Jarecki, and H. Krawczyk. Robust and efficient sharing of RSA functions. *J. Cryptology*, 13(2):273–300, 2000.

[13] C. Gentry and A. Silverberg. Hierarchical ID-based cryptography. In *Proceedings of Asiacrypt 2002*, volume 2501 of *LNCS*, pages 548–66. Springer-Verlag, 2002.

[14] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2):281–308, 1988.

[15] J. Horwitz and B. Lynn. Toward hierarchical identity-based encryption. In *Proceedings of Eurocrypt 2002*, volume 2332 of *LNCS*, pages 466–81. Springer-Verlag, 2002.

[16] A. Joux. A one round protocol for tripartite Diffie-Hellman. In *Proceedings of ANTS IV*, volume 1838 of *LNCS*, pages 385–94. Springer-Verlag, 2000.

[17] A. Joux and K. Nguyen. Separating Decision Diffie-Hellman from Diffie-Hellman in Cryptographic Groups. Cryptology ePrint Archive, Report 2001/003, 2001. `http://eprint.iacr.org/`.

[18] S. Kent, C. Lynn, and K. Seo. Secure border gateway protocol (Secure-BGP). *IEEE J. Selected Areas in Comm.*, 18(4):582–92, April 2000.

[19] A. Lysyanskaya. Unique signatures and verifiable random functions from the DH-DDH separation. In *Proceedings of Crypto 2002*, volume 2442 of *LNCS*, pages 597–612. Springer-Verlag, 2002.

[20] S. Micali, K. Ohta, and L. Reyzin. Accountable-subgroup multisignatures (extended abstract). In *Proceedings of CCS 2001*, pages 245–54. ACM Press, 2001.

[21] S. Micali and R. Rivest. Transitive signature schemes. In *Proceedings of RSA 2002*, volume 2271 of *LNCS*, pages 236–43. Springer-Verlag, 2002.

[22] A. Miyaji, M. Nakabayashi, and S. Takano. New explicit conditions of elliptic curve traces for FR-reduction. *IEICE Trans. Fundamentals*, E84-A(5):1234–43, May 2001.

[23] M. Naor. Deniable ring authentication. In *Proceedings of Crypto 2002*, volume 2442 of *LNCS*, pages 481–98. Springer-Verlag, 2002.

[24] K. Ohta and T. Okamoto. Multisignature schemes secure against active insider attacks. *IEICE Trans. Fundamentals*, E82-A(1):21–31, 1999.

[25] T. Okamoto. A digital multisignature scheme using bijective public-key cryptosystems. *ACM Trans. Computer Systems*, 6(4):432–441, 1998.

[26] T. Okamoto and D. Pointcheval. The gap problems: A new class of problems for the security of cryptographic primitives. In *Proceedings of PKC 2001*, volume 1992 of *LNCS*, pages 104–118. Springer-Verlag, 2001.

[27] G. Poupard and J. Stern. Fair encryption of RSA keys. In *Proceedings of Eurocrypt 2000*, volume 1807 of *LNCS*, pages 172–89. Springer-Verlag, 2000.

[28] R. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In *Proceedings of Asiacrypt 2001*, volume 2248 of *LNCS*, pages 552–65. Springer-Verlag, 2001.

[29] F. Zhang and K. Kim. ID-based blind signature and ring signature from pairings. In *Proceedings of Asiacrypt 2002*, volume 2501 of *LNCS*, pages 533–47. Springer-Verlag, 2002.

Short Signatures Without Random Oracles

Dan Boneh
dabo@cs.stanford.edu

Xavier Boyen
xb@boyen.org

Abstract

We describe a short signature scheme which is existentially unforgeable under a chosen message attack without using random oracles. The security of our scheme depends on a new complexity assumption we call the *Strong Diffie-Hellman* assumption. This assumption has similar properties to the Strong RSA assumption, hence the name. Strong RSA was previously used to construct signature schemes without random oracles. However, signatures generated by our scheme are much shorter and simpler than signatures from schemes based on Strong RSA. Furthermore, our scheme provides a limited form of message recovery.

1 Introduction

Boneh, Lynn, and Shacham (BLS) [BLS01] recently proposed a short digital signature scheme where signatures are about half the size of DSA signatures with the same level of security. Security is based on the Computational Diffie-Hellman (CDH) assumption on certain elliptic curves. The scheme is shown to be existentially unforgeable under a chosen message attack in the random oracle model [BR93].

In this paper we describe a signature scheme where signatures are almost as short as BLS signatures, but whose security does not require random oracles. We prove security of our scheme using a complexity assumption we call the Strong Diffie-Hellman assumption, or SDH for short. Roughly speaking, the q-SDH assumption in a group $\mathbb{G}$ of prime order p states that the following problem is intractable: given $g, g^x, g^{(x^2)}, \ldots, g^{(x^q)} \in \mathbb{G}$ as input, output a pair $(c,\ g^{1/(x+c)})$ where $c \in \mathbb{Z}_p^*$. Precise definitions are given in Section 2.3. Using this assumption we construct a signature scheme that is existentially unforgeable under a chosen message attack *without using random oracles*.

Currently, the most practical signature schemes secure without random oracles [GHR99, CS00] are based on the Strong RSA assumption (given an RSA modulus N and $s \in \mathbb{Z}_N^*$ it is difficult to construct a non-trivial pair $(c, s^{1/c})$ where $c \in \mathbb{Z}$). Roughly speaking, what makes Strong RSA so useful for constructing secure signature schemes is the following property: given a Strong RSA problem instance (N, s) it is possible to construct a new instance (N, s') with q known solutions $(c_i, (s')^{1/c_i})$, where the construction of any other solution $(c, (s')^{1/c})$ makes it possible to solve the original problem instance. This property provides a way to prove security against a chosen message attack. In Section 3.1 we show that the q-SDH problem has a similar property. Hence, q-SDH may be viewed as a discrete logarithm analogue of the Strong RSA assumption. We believe that the properties of q-SDH make it a useful tool for constructing cryptographic systems and we expect to see many other systems based on it.

To gain some confidence in the q-SDH assumption we provide in Section 5 a lower bound on the computational complexity of solving the q-SDH problem in a generic group model. This shows

that no generic attack on q-SDH is possible. Mitsunari, Sakai, and Kasahara [MSK02] previously used a weaker variant of the q-SDH assumption to construct a traitor tracing scheme. The ideas in their paper are nice, and we use some of them here. However, their application to tracing traitors appears to be insecure [TSNZ03].

We present our secure signature scheme in Section 3 and prove its security against existential forgery under chosen message attack. The resulting signatures are as short as DSA signatures, but are provably secure in the absence of random oracles. Our signatures also support limited message recovery, which makes it possible to further reduce the total length of a message/signature pair. In Section 4 we show that with random oracles the q-SDH assumption gives even shorter signatures. A related system using random oracles was recently described by Zhang et al. [ZSNS04].

We refer to [BLS01] for applications of short signatures. We only mention that short digital signatures are needed in environments with stringent bandwidth constraints, such as bar-coded digital signatures on postage stamps [NS00, PV00]. We also note that Patarin et al. [PCG01, CDF03] construct short signatures whose security depends on the Hidden Field Equation (HFE) problem.

2 Preliminaries

Before presenting our results we briefly review two notions of security for signature schemes, review the definition for groups equipped with a bilinear map, and precisely state the q-SDH assumption.

2.1 Secure Signature Schemes

A signature scheme is made up of three algorithms, *KeyGen*, *Sign*, and *Verify*, for generating keys, signing, and verifying signatures, respectively.

Strong Existential Unforgeability

The standard notion of security for a signature scheme is called existential unforgeability under a chosen message attack [GMR88]. We consider a slightly stronger notion of security, called strong existential unforgeability [ADR02], which is defined using the following game between a challenger and an adversary $\mathcal{A}$:

Setup: The challenger runs algorithm *KeyGen* to obtain a public key PK and a private key SK. The adversary $\mathcal{A}$ is given PK.

Queries: Proceeding adaptively, $\mathcal{A}$ requests signatures on at most q_S messages of his choice $M_1, \ldots, M_{q_s} \in \{0,1\}^*$, under PK. The challenger responds to each query with a signature $\sigma_i = \mathit{Sign}(SK, M_i)$.

Output: Eventually, $\mathcal{A}$ outputs a pair (M, σ) and wins the game if
(1) (M, σ) is not any of $(M_1, \sigma_1), \ldots, (M_{q_s}, \sigma_{q_s})$, and
(2) $\mathit{Verify}(PK, M, \sigma) = \texttt{valid}$.

We define $\mathsf{Adv}\,\mathsf{Sig}_{\mathcal{A}}$ to be the probability that $\mathcal{A}$ wins in the above game, taken over the coin tosses made by $\mathcal{A}$ and the challenger.

Definition 2.1. A forger $\mathcal{A}$ (t, q_S, ϵ)-breaks a signature scheme if $\mathcal{A}$ runs in time at most t, $\mathcal{A}$ makes at most q_S signature queries, and $\mathsf{Adv}\,\mathsf{Sig}_{\mathcal{A}}$ is at least ϵ. A signature scheme is (t, q_S, ϵ)-existentially unforgeable under an adaptive chosen message attack if no forger (t, q_S, ϵ)-breaks it.

When proving security in the random oracle model we add a fourth parameter q_H denoting an upper bound on the number of queries that the adversary $\mathcal{A}$ makes to the random oracle.

We note that the definition above captures a stronger version of existential unforgeability than the standard one: we require that the adversary cannot even generate a new signature on a previously signed message. This property is required for some applications [ADR02, Sah99, CHK04]. All our signature schemes satisfy this stronger security notion.

Weak Chosen Message Attacks

We will also use a weaker notion of security which we call existential unforgeability under a weak chosen message attack. Here we require that the adversary submit all signature queries before seeing the public key. This notion is defined using the following game between a challenger and an adversary $\mathcal{A}$:

Query: $\mathcal{A}$ sends the challenger a list of q_S messages $M_1, \ldots, M_{q_s} \in \{0,1\}^*$.

Response: The challenger runs algorithm *KeyGen* to generate a public key PK and private key SK. Next, the challenger generates signatures $\sigma_i = \mathit{Sign}(SK, M_i)$ for $i = 1, \ldots, q_S$. The challenger then gives $\mathcal{A}$ the public key PK and the q_S signatures $\sigma_1, \ldots, \sigma_{q_s}$.

Output: Algorithm $\mathcal{A}$ outputs a pair (M, σ) and wins the game if
(1) M is not any of $M_1, \ldots, M_{q_s}$, and
(2) $\mathit{Verify}(PK, M, \sigma) = \texttt{valid}$.

We define $\mathsf{Adv}\,\mathsf{W\text{-}Sig}_{\mathcal{A}}$ to be the probability that $\mathcal{A}$ wins in the above game, taken over the coin tosses of $\mathcal{A}$ and the challenger.

Definition 2.2. A forger $\mathcal{A}$ (t, q_S, ϵ)-weakly breaks a signature scheme if $\mathcal{A}$ runs in time at most t, $\mathcal{A}$ makes at most q_S signature queries, and $\mathsf{Adv}\,\mathsf{W\text{-}Sig}_{\mathcal{A}}$ is at least ϵ. A signature scheme is (t, q_S, ϵ)-existentially unforgeable under a weak chosen message attack if no forger (t, q_S, ϵ)-weakly breaks it.

2.2 Bilinear Groups

Signature verification in our scheme requires a bilinear map. We briefly review the necessary facts about bilinear maps and bilinear map groups. We follow the notation in [BLS01]:

1. $\mathbb{G}_1$ and $\mathbb{G}_2$ are two (multiplicative) cyclic groups of prime order p;
2. g_1 is a generator of $\mathbb{G}_1$ and g_2 is a generator of $\mathbb{G}_2$;
3. ψ is an isomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$, with $\psi(g_2) = g_1$; and
4. e is a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.

For simplicity one can set $\mathbb{G}_1 = \mathbb{G}_2$. However, as in [BLS01], we allow for the more general case where $\mathbb{G}_1 \neq \mathbb{G}_2$ so that we can take advantage of certain families of elliptic curves to obtain short signatures. Specifically, elements of $\mathbb{G}_1$ have a short representation whereas elements of $\mathbb{G}_2$ may not. The proofs of security require an efficiently computable isomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$. When $\mathbb{G}_1 = \mathbb{G}_2$ and $g_1 = g_2$ one could take ψ to be the identity map. On elliptic curves we can use the trace map as ψ.

Let thus $\mathbb{G}_1$ and $\mathbb{G}_2$ be two groups as above, with an additional group $\mathbb{G}_T$ such that $|\mathbb{G}_1| = |\mathbb{G}_2| = |\mathbb{G}_T|$. A bilinear map is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with the following properties:

1. Bilinear: for all $u \in \mathbb{G}_1, v \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}$, $e(u^a, v^b) = e(u, v)^{ab}$.
2. Non-degenerate: $e(g_1, g_2) \neq 1$.

We say that $(\mathbb{G}_1, \mathbb{G}_2)$ are bilinear groups if there exists a group $\mathbb{G}_T$, an isomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$, and a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ as above, and e, ψ, and the group action in $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ can be computed efficiently.

Joux and Nguyen [JN01] showed that an efficiently computable bilinear map e provides an algorithm for solving the Decision Diffie-Hellman problem (DDH). Our results can be stated using a generic algorithm for DDH. Nevertheless, for the sake of concreteness we instead describe our results by directly referring to the bilinear map.

2.3 The Strong Diffie-Hellman Assumption

Before describing the new signature schemes, we first state precisely the hardness assumption on which they are based. Let $\mathbb{G}_1, \mathbb{G}_2$ be two cyclic groups of prime order p, where possibly $\mathbb{G}_1 = \mathbb{G}_2$. Let g_1 be a generator of $\mathbb{G}_1$ and g_2 a generator of $\mathbb{G}_2$ such that $g_1 = \psi(g_2)$.

q-Strong Diffie-Hellman Problem. The q-SDH problem in $(\mathbb{G}_1, \mathbb{G}_2)$ is defined as follows: given a $(q+2)$-tuple $(g_1, g_2, g_2^x, g_2^{(x^2)}, \ldots, g_2^{(x^q)})$ as input where as above $g_1 = \psi(g_2)$, output a pair $(c,\, g_1^{1/(x+c)})$ where $c \in \mathbb{Z}_p^*$. An algorithm $\mathcal{A}$ has advantage ϵ in solving q-SDH in $(\mathbb{G}_1, \mathbb{G}_2)$ if

$$\Pr\left[\mathcal{A}(g_1, g_2, g_2^x, \ldots, g_2^{(x^q)}) = (c,\, g_1^{\frac{1}{x+c}})\right] \geq \epsilon$$

where the probability is over the random choice of generator $g_2 \in \mathbb{G}_2$ with $g_1 = \psi(g_2)$, the random choice of x in $\mathbb{Z}_p^*$, and the random bits consumed by $\mathcal{A}$.

Definition 2.3. We say that the (q, t, ϵ)-SDH assumption holds in $(\mathbb{G}_1, \mathbb{G}_2)$ if no t-time algorithm has advantage at least ϵ in solving the q-SDH problem in $(\mathbb{G}_1, \mathbb{G}_2)$.

Occasionally we drop the t and ϵ and refer to the q-SDH assumption rather than the (q, t, ϵ)-SDH assumption. As we will see in the next section the q-SDH assumption has similar properties to the Strong RSA problem and we therefore view q-SDH as a discrete logarithm analogue of the Strong RSA assumption.

To provide some confidence in the q-SDH assumption, we prove in Section 5 a lower bound on the complexity of solving the q-SDH problem in a generic group. Furthermore, we note that the Strong Diffie-Hellman problem has a simple random self-reduction in $(\mathbb{G}_1, \mathbb{G}_2)$.

A weaker version of the q-SDH assumption was previously used by Mitsunari, Sakai, and Kasahara [MSK02] to construct a traitor tracing system (see [TSNZ03] for an analysis). Using our notation, their version of the assumption requires Algorithm $\mathcal{A}$ to output $g_1^{1/(x+c)}$ for a *given input value* c. In the assumption above we allow $\mathcal{A}$ to choose c. When c is pre-specified the q-SDH problem is equivalent to the following problem: given $(g_1, g_2, g_2^x, g_2^{x^2}, \ldots, g_2^{x^q})$ output $g_1^{1/x}$. We note that when $\mathcal{A}$ is allowed to choose c no such equivalence is known.

3 Short Signatures Without Random Oracles

We now construct a fully secure short signature scheme in the standard model using the q-SDH assumption. We consider this to be the main result of the paper.

Let $(\mathbb{G}_1, \mathbb{G}_2)$ be bilinear groups where $|\mathbb{G}_1| = |\mathbb{G}_2| = p$ for some prime p. For the moment we assume that the messages m to be signed are elements in $\mathbb{Z}_p^*$, but as we mention in Section 3.5, the domain can be extended to all of $\{0,1\}^*$ using a collision resistant hash function $H : \{0,1\}^* \to \mathbb{Z}_p^*$.

Key generation: Pick a random generator $g_2 \in G_2$ and set $g_1 = \psi(g_2)$. Pick random $x, y \stackrel{R}{\leftarrow} \mathbb{Z}_p^*$, and compute $u \leftarrow g_2^x \in \mathbb{G}_2$ and $v \leftarrow g_2^y \in \mathbb{G}_2$. Also compute $z \leftarrow e(g_1, g_2) \in \mathbb{G}_T$. The public key is (g_1, g_2, u, v, z). The secret key is (x, y).

Signing: Given a secret key $x, y \in \mathbb{Z}_p^*$ and a message $m \in \mathbb{Z}_p^*$, pick a random $r \in \mathbb{Z}_p^*$ and compute $\sigma \leftarrow g_1^{1/(x+m+yr)} \in \mathbb{G}_1$. Here $1/(x+m+yr)$ is computed modulo p. In the unlikely event that $x + m + yr = 0$ we try again with a different random r. The signature is (σ, r).

Verification: Given a public key (g_1, g_2, u, v, z), a message $m \in \mathbb{Z}_p^*$, and a signature (σ, r), verify that

$$e(\sigma,\ u \cdot g_2^m \cdot v^r) = z$$

If the equality holds the result is `valid`; otherwise the result is `invalid`.

Public Key Integrity. Observe that the g_1 and z components of the public key can be computed from other parts of the public key and are thus redundant. They are included in the public key for efficiency reasons, in order to dispense the verifier from performing these computations. It is up to the Certifying Authority (CA) to verify that the relations $g_1 = \psi(g_2)$ and $z = e(g_1, g_2)$ hold before issuing a certificate. Alternatively, the verifier can perform these checks when verifying the certificate for a given public-key. Since this is a one-time check we do not explicitly include it in the verification algorithm.

Signature Length. A signature contains two elements (σ, r), each of length approximately $\log_2(p)$ bits, therefore the total signature length is approximately $2\log_2(p)$. When using the elliptic curves described in [BLS01] we obtain a signature whose length is approximately the same as a DSA signature with the same security, but which is provably existentially unforgeable under a chosen message attack without the random oracle model.

Performance. Key and signature generation times are comparable to BLS signatures. Verification time is faster since verification requires only one pairing and one multi-exponentiation. The value $z = e(g_1, g_2)$ only needs to be computed (or verified) at certification time. In comparison, BLS signature verification requires two pairing computations. Since exponentiation tends to be significantly faster than pairing, signature verification is faster than in the BLS system.

Security. The following theorem shows that the scheme above is existentially unforgeable in the strong sense under chosen message attacks, provided that the q-SDH assumption holds in $(\mathbb{G}_1, \mathbb{G}_2)$.

Theorem 3.1. *Suppose the (q, t', ϵ')-SDH assumption holds in $(\mathbb{G}_1, \mathbb{G}_2)$. Then the signature scheme above is (t, q_S, ϵ)-secure against existential forgery under a chosen message attack provided that*

$$q_S < q\ , \qquad \epsilon \geq 2(\epsilon' + q_S/p) \approx 2\epsilon' \qquad \textit{and} \qquad t \leq t' - \Theta(q^2 T)$$

where T is the maximum time for an exponentiation in G_1 and G_2.

Proof. We prove the theorem using two lemmas. In Lemma 3.2, we first describe a simplified signature scheme and prove its existential unforgeability against *weak* chosen message attacks under the q-SDH assumption. In Lemma 3.3, we then show that the security of the weak scheme implies the security of the full scheme. From these results (Lemmas 3.2 and 3.3), Theorem 3.1 follows easily. We present the proof in two steps since the construction used to prove Lemma 3.2 will be used later on in the paper. □

3.1 A Weakly Secure Short Signature Scheme

We first show how the q-SDH assumption can be used to construct an existentially unforgeable scheme under a *weak* chosen message attack. This construction demonstrates the main properties of the q-SDH assumption. In the next section we show that the security of this weak scheme implies the security of the full scheme above.

The weakly secure short signature scheme is as follows. As before, let $(\mathbb{G}_1, \mathbb{G}_2)$ be bilinear groups where $|\mathbb{G}_1| = |\mathbb{G}_2| = p$ for some prime p. For the moment we assume that the messages m to be signed are elements in $\mathbb{Z}_p^*$.

Key generation: Pick a random generator $g_2 \in G_2$ and set $g_1 = \psi(g_2)$. Pick random $x \xleftarrow{R} \mathbb{Z}_p^*$, and compute $v \leftarrow g_2^x \in \mathbb{G}_2$ and $z \leftarrow e(g_1, g_2) \in \mathbb{G}_T$. The public key is (g_1, g_2, v, z). The secret key is x.

Signing: Given a secret key $x \in \mathbb{Z}_p^*$ and a message $m \in \mathbb{Z}_p^*$, output the signature $\sigma \leftarrow g_1^{1/(x+m)} \in \mathbb{G}_1$. Here $1/(x+m)$ is computed modulo p. By convention in this context we define $1/0$ to be 0 so that in the unlikely event that $x + m = 0$ we have $\sigma \leftarrow 1$.

Verification: Given a public key (g_1, g_2, v, z), a message $m \in \mathbb{Z}_p^*$, and a signature $\sigma \in \mathbb{G}_1$, verify that

$$e(\sigma, v \cdot g_2^m) = z$$

If equality holds output `valid`. If $\sigma = 1$ and $v \cdot g_2^m = 1$ output `valid`.
Otherwise, output `invalid`.

We show that the basic signature scheme above is existentially unforgeable under a *weak* chosen message attack. The proof of the following lemma uses a similar method to the proof of Theorem 3.5 of Mitsunari et al. [MSK02].

Lemma 3.2. *Suppose the (q, t', ϵ)-SDH assumption holds in $(\mathbb{G}_1, \mathbb{G}_2)$. Then the basic signature scheme above is (t, q_S, ϵ)-secure against existential forgery under a weak chosen message attack provided that*

$$q_S < q \qquad \text{and} \qquad t \leq t' - \Theta(q^2 T)$$

where T is the maximum time for an exponentiation in G_1 and G_2.

Proof. Assume $\mathcal{A}$ is a forger that (t, q_S, ϵ)-breaks the signature scheme. We construct an algorithm $\mathcal{B}$ that, by interacting with $\mathcal{A}$, solves the q-SDH problem in time t' with advantage ϵ. Algorithm $\mathcal{B}$ is given a random instance $(g_1, g_2, A_1, \ldots, A_q)$ of the q-SDH problem, where $A_i = g_2^{(x^i)} \in \mathbb{G}_2$ for $i = 1, \ldots, q$ and for some unknown $x \in \mathbb{Z}_p^*$. For convenience we set $A_0 = g_2$. Algorithm $\mathcal{B}$'s goal is to produce a pair $(c, g_1^{1/(x+c)})$ for some $c \in \mathbb{Z}_p^*$. Algorithm $\mathcal{B}$ does so by interacting with the forger $\mathcal{A}$ as follows:

Query: Algorithm $\mathcal{A}$ outputs a list of distinct q_S messages $m_1, \ldots, m_{q_s} \in \mathbb{Z}_p^*$, where $q_S < q$. Since $\mathcal{A}$ must reveal its queries up front, we may assume that $\mathcal{A}$ outputs exactly $q - 1$ messages to be signed (if the actual number is less, we can always virtually reduce the value of q so that $q = q_S + 1$).

Response: $\mathcal{B}$ must respond with a public key and signatures on the $q - 1$ messages from $\mathcal{A}$. Let $f(y)$ be the polynomial $f(y) = \prod_{i=1}^{q-1}(y + m_i)$. Expand $f(y)$ and write $f(y) = \sum_{i=0}^{q-1} \alpha_i y^i$

where $\alpha_0, \ldots, \alpha_{q-1} \in \mathbb{Z}_p$ are the coefficients of the polynomial $f(y)$. Compute:

$$g_2' \leftarrow \prod_{i=0}^{q-1} A_i^{\alpha_i} = g_2^{f(x)} \quad \text{and} \quad h \leftarrow \prod_{i=1}^{q} A_i^{\alpha_{i-1}} = g_2^{xf(x)} = (g_2')^x$$

Also, let $g_1' = \psi(g_2')$ and $z' = e(g_1', g_2')$. The public key given to $\mathcal{A}$ is (g_1', g_2', h, z'), which has the correct distribution. Note that we may assume that $f(x) \neq 0$ since, otherwise, $x = -m_i$ for some i which means that $\mathcal{B}$ just obtained the secret key x.

Next, for each $i = 1, \ldots q-1$, Algorithm $\mathcal{B}$ must generate a signature σ_i on m_i. To do so, let $f_i(y)$ be the polynomial $f_i(y) = f(y)/(y+m_i) = \prod_{j=1, j\neq i}^{q-1}(y+m_j)$. As before, we expand f_i and write $f_i(y) = \sum_{j=0}^{q-2} \beta_j y^j$. Compute

$$S_i \leftarrow \prod_{j=0}^{q-2} A_j^{\beta_j} = g_2^{f_i(x)} = (g_2')^{1/(x+m_i)} \in \mathbb{G}_2$$

Observe that $\sigma_i = \psi(S_i) \in \mathbb{G}_1$ is a valid signature on m under the public key (g_1', g_2', h, z'). Algorithm $\mathcal{B}$ gives $\mathcal{A}$ the $q-1$ signatures $\sigma_1, \ldots, \sigma_{q-1}$.

Output: Algorithm $\mathcal{A}$ returns a forgery (m_*, σ_*) such that $\sigma_* \in \mathbb{G}_1$ is a valid signature on $m_* \in \mathbb{Z}_p^*$ and $m_* \notin \{m_1 \ldots, m_{q-1}\}$ since there is only one valid signature per message. In other words, $e(\sigma_*, h \cdot (g_2')^{m_*}) = e(g_1', g_2')$. Since $h = (g_2')^x$ we have that $e(\sigma_*, (g_2')^{x+m_*}) = e(g_1', g_2')$ and therefore

$$\sigma_* = (g_1')^{1/(x+m_*)} = (g_1)^{f(x)/(x+m_*)} \tag{1}$$

Using long division we write the polynomial f as $f(y) = \gamma(y)(y+m_*)+\gamma_{-1}$ for some polynomial $\gamma(y) = \sum_{i=0}^{q-2} \gamma_i y^i$ and some $\gamma_{-1} \in \mathbb{Z}_p$. Then the rational fraction $f(y)/(y+m_*)$ in the exponent on the right side of Equation (1) can be written as

$$f(y)/(y+m_*) = \frac{\gamma_{-1}}{y+m_*} + \sum_{i=0}^{q-2} \gamma_i y^i \qquad \text{and hence} \qquad \sigma_* = g_1^{\frac{\gamma_{-1}}{x+m_*} + \sum_{i=0}^{q-2} \gamma_i x^i}$$

Note that $\gamma_{-1} \neq 0$, since $f(y) = \prod_{i=1}^{q-1}(y+m_i)$ and $m_* \notin \{m_1, \ldots, m_{q-1}\}$, as thus $(y+m_*)$ does not divide $f(y)$. Then algorithm $\mathcal{B}$ computes

$$w \leftarrow \left(\sigma_* \cdot \prod_{i=0}^{q-2} \psi(A_i)^{-\gamma_i}\right)^{1/\gamma_{-1}} = \left(g_1^{\frac{\gamma_{-1}}{x+m_*}} \cdot g_1^{\sum_{i=0}^{q-2} \gamma_i x^i} \cdot \prod_{i=0}^{q-2} g_1^{-\gamma_i x_i}\right)^{1/\gamma_{-1}} = g_1^{1/(x+m_*)}$$

and returns (m_*, w) as the solution to the q-SDH instance.

The claimed bounds are obvious by construction of the reduction. □

3.2 From Weak Security To Full Security

We now present a reduction from the security of the basic scheme of Lemma 3.2 to the security of the full signature scheme described at the onset of Section 3. This will complete the proof of Theorem 3.1.

Lemma 3.3. *Suppose that the basic signature scheme of Lemma 3.2 is (t', q_S, ϵ')-weakly secure. Then the full signature scheme is (t, q_S, ϵ)-secure against existential forgery under a chosen message attack provided that*

$$\epsilon \geq 2(\epsilon' + q_S/p) \approx 2\epsilon' \qquad \text{and} \qquad t \leq t' - \Theta(q_S T)$$

where T is the maximum time for an exponentiation in G_1 and G_2.

We first give some intuition for the proof. Suppose $\mathcal{A}$ is a forger for the full scheme under a chosen message attack. We build a forger $\mathcal{B}$ for the weak scheme under a weak chosen message attack. Forger $\mathcal{B}$ starts by requesting signatures on random messages $w_1, \ldots, w_q \in \mathbb{Z}_p^*$. In response, it is given a public key (g_1, g_2, u, z) and signatures $\sigma_1, \ldots, \sigma_{q_s} \in \mathbb{G}_1$ for the weak scheme. In principle, $\mathcal{B}$ could create a public key for the full scheme by picking a random $y \in \mathbb{Z}_p^*$ and giving $\mathcal{A}$ the public key (g_1, g_2, u, g_2^y, z). Now, when $\mathcal{A}$ issues a chosen message query for a message $m_i \in \mathbb{Z}_p^*$, forger $\mathcal{B}$ could choose an $r_i \in \mathbb{Z}_p$ such that $m_i + yr_i$ maps to w_i. Then (σ_i, r_i) is a valid signature on m_i for the full scheme and hence a proper response to $\mathcal{A}$'s chosen message query. Eventually, $\mathcal{A}$ outputs a forgery (m_*, σ_*, r_*). Then $(m_* + yr_*, \sigma_*)$ is a valid message/signature pair for the weak scheme. In principle, $\mathcal{B}$ could output this pair as its existential forgery on the weak scheme. The problem is that $m_* + yr_*$ might be in $\{w_1, \ldots, w_{q_s}\}$ in which case $(m_* + yr_*, \sigma_*)$ is an invalid existential forgery. Dealing with this case complicates the proof and forces us to consider two types of adversaries. The full proof is given below.

Proof of Lemma 3.3. Assume $\mathcal{A}$ is a forger that (t, q_S, ϵ)-breaks the full signature scheme. We construct an algorithm $\mathcal{B}$ that $(t',\ q_S,\ \epsilon/2 - q_S/p)$-weakly breaks the basic signature scheme of Lemma 3.2.

Before describing Algorithm $\mathcal{B}$ we distinguish between two types of forgers that $\mathcal{A}$ can emulate. Let (h_1, h_2, u, v, z) be the public key given to forger $\mathcal{A}$ where $u = g_2^x$ and $v = g_2^y$. Suppose $\mathcal{A}$ asks for signatures on messages $m_1, \ldots, m_{q_s} \in \mathbb{Z}_p^*$ and is given signatures (σ_i, r_i) for $i = 1, \ldots, q_S$ on these messages. Let $w_i = m_i + yr_i$ and let (m_*, σ_*, r_*) be the forgery produced by $\mathcal{A}$. We distinguish between two types of forgers:

Type-1 forger: a forger that either
(i) makes a signature query for the message $m = -x$, or
(ii) outputs a forgery where $m_* + yr_* \notin \{w_1, \ldots, w_{q_s}\}$.

Type-2 forger: a forger that both
(i) never makes a signature query for the message $m = -x$, and
(ii) outputs a forgery where $m_* + yr_* = w_i$ for some $i \in \{1, \ldots, q_S\}$.

We show that either forger can be used to forge signatures for the weak signature scheme of Lemma 3.2. However, the reduction works differently for each forger type. Therefore, initially $\mathcal{B}$ will choose a random bit $c_{\text{mode}} \in \{1, 2\}$ that indicates its guess for the type of forger that $\mathcal{A}$ will emulate. The simulation proceeds differently for each mode.

We are now ready to describe Algorithm $\mathcal{B}$. It produces a forgery for the signature scheme of Lemma 3.2 as follows:

Setup: Algorithm $\mathcal{B}$ first picks a random bit $c_{\text{mode}} \in \{1, 2\}$. Next, $\mathcal{B}$ sends to its own challenger a list of q_S random messages $w_1, \ldots, w_{q_s} \in \mathbb{Z}_p^*$ for which it requests a signature. The challenger responds with a valid public key (g_1, g_2, u, z) and signatures $\sigma_1, \ldots, \sigma_{q_s} \in \mathbb{G}_1$ on these messages. We know that $e(\sigma_i, g_2^{w_i} u) = e(g_1, g_2) = z$ for all $i = 1, \ldots, q_S$. Then:

- (If $c_{\text{mode}} = 1$). $\mathcal{B}$ picks a random $y \in \mathbb{Z}_p^*$ and gives $\mathcal{A}$ the public key $PK_1 = (g_1, g_2, u, g_2^y, z)$.

• (If $c_{\text{mode}} = 2$). $\mathcal{B}$ picks a random $x \in \mathbb{Z}_p^*$ and gives $\mathcal{A}$ the public key $PK_2 = (g_1, g_2, g_2^x, u, z)$.

In either case, we note that $\mathcal{B}$ provides the adversary $\mathcal{A}$ with a valid public key (g_1, g_2, U, V, z).

Signature queries: The forger $\mathcal{A}$ can issue up to q_S signature queries in an adaptive fashion. In order to respond, $\mathcal{B}$ maintains a list H-list of tuples (m_i, r_i, W_i) and a query counter ℓ which is initially set to 0. Upon receiving a signature query for m, Algorithm $\mathcal{B}$ increments ℓ by one. Then:

• (If $c_{\text{mode}} = 1$). Check if $g_2^{-m} = u$. If so, then $\mathcal{B}$ just obtained the private key for the public key (g_1, g_2, u, z) it was given, which allows it to forge the signature on any message of its choice. At this point $\mathcal{B}$ successfully terminates the simulation.
Otherwise, set $r_\ell = (w_\ell - m)/y \in \mathbb{Z}_p^*$. In the very unlikely event that $r_\ell = 0$, Algorithm $\mathcal{B}$ reports failure and aborts. Otherwise, Algorithm $\mathcal{B}$ gives $\mathcal{A}$ the signature (σ_ℓ, r_ℓ). This is a valid signature on m under PK_1 since r_ℓ is uniform in $\mathbb{Z}_p^*$ and

$$e(\sigma_\ell,\ U \cdot g_2^m \cdot V^{r_\ell}) = e(\sigma_\ell,\ u \cdot g_2^m \cdot g_2^{yr_\ell}) = e(\sigma_\ell, u \cdot g_2^{w_\ell}) = e(g_1, g_2) = z$$

• (If $c_{\text{mode}} = 2$). Set $r_\ell = (x+m)/w_\ell \in \mathbb{Z}_p^*$. If $r_\ell = 0$, Algorithm $\mathcal{B}$ reports failure and aborts. Otherwise, give $\mathcal{A}$ the signature $(\sigma_\ell^{1/r_\ell}, r_\ell)$. This is a valid signature on m under PK_2 since r_ℓ is uniform in $\mathbb{Z}_p^*$ and

$$e(\sigma_\ell^{1/r_\ell},\ U \cdot g_2^m \cdot V^{r_\ell}) = e(\sigma_\ell^{1/r_\ell},\ g_2^x \cdot g_2^m \cdot u^{r_\ell}) = e(\sigma_\ell,\ g_2^{w_\ell} u) = e(g_1, g_2) = z$$

In either case if $\mathcal{B}$ does not abort it responds with a valid signature on m.
In either case Algorithm $\mathcal{B}$ adds the tuple $(m, r_\ell, g_2^m V^{r_\ell})$ to the H-list.

Output: Eventually, $\mathcal{A}$ returns a forgery (m_*, σ_*, r_*), where (σ_*, r_*) is a valid forgery distinct from any previously given signature on message m_*. Note that by adding dummy queries as necessary, we may assume that $\mathcal{A}$ made exactly q_S signature queries. Let $W_* \leftarrow g_2^{m_*} V^{r_*}$. Algorithm $\mathcal{B}$ searches the H-list for a tuple whose rightmost component is equal to W_*. There are two possibilities:

Type-1 forgery: No tuple of the form $(\cdot, \cdot, W_*)$ appears on the H-list.

Type-2 forgery: The H-list contains at least one tuple (m_j, r_j, W_j) such that $W_j = W_*$.

Let $b_{\text{type}} \leftarrow 1$ if $\mathcal{A}$ produced a type-1 forgery, or $\mathcal{A}$ made a signature query for a message m such that $g_2^{-m} = U$. In all other cases, set $b_{\text{type}} \leftarrow 2$. If $b_{\text{type}} \neq c_{\text{mode}}$ then $\mathcal{B}$ reports failure and aborts. Otherwise, $\mathcal{B}$ outputs an existential forgery on the basic signature scheme as follows:

• (If $c_{\text{mode}} = b_{\text{type}} = 1$). If $\mathcal{A}$ made a signature query for a message m such that $g_2^{-m} = U$ then $\mathcal{B}$ is already done. Therefore, we assume $\mathcal{A}$ produced a type-1 forgery. Since the forgery is valid, we have

$$e(g_1, g_2) = e(\sigma_*,\ U \cdot g_2^{m_*} \cdot V^{r_*}) = e(\sigma_*,\ u \cdot g_2^{m_* + yr_*})$$

Let $w_* = m_* + yr_*$. It follows that (w_*, σ_*) is a valid message/signature pair in the basic signature scheme. Furthermore, it is a valid existential forgery for the basic scheme since in

a type-1 forgery Algorithm $\mathcal{B}$ did not request a signature on the message $w_* \in \mathbb{Z}_p^*$. Indeed, $\mathcal{B}$ only requested signatures on messages $w_j = m_j + yr_j$ where $(m_j, r_j, g_2^{w_j})$ is a tuple in the H-list, but $g_2^{w_*}$ is not equal to any $g_2^{w_j}$ on the H-list. Algorithm $\mathcal{B}$ outputs (w_*, σ_*) as the required existential forgery.

• (If $c_{\text{mode}} = b_{\text{type}} = 2$). Let (m_j, r_j, W_j) be a tuple on the H-list where $W_j = W_*$. Since $V = u$ we know that $g_2^{m_j} u^{r_j} = g_2^{m_*} u^{r_*}$. Write $u = g_2^{\tau}$ for some $\tau \in \mathbb{Z}_p^*$ so that $m_j + \tau r_j = m_* + \tau r_*$. We know that $(m_j, r_j) \neq (m_*, r_*)$, otherwise the forgery would be identical to a previously given signature on the query message m_j. Since $g_2^{m_j} u^{r_j} = g_2^{m_*} u^{r_*}$ it follows that $m_j \neq m_*$ and $r_j \neq r_*$. Therefore, $\tau = (m_* - m_j)/(r_j - r_*) \in \mathbb{Z}_p^*$. Hence, $\mathcal{B}$ just recovered the private key, τ, for the public key (g_1, g_2, u, z) it was given. Algorithm $\mathcal{B}$ can now forge a signature on any message of its choice.

This completes the description of Algorithm $\mathcal{B}$. A standard argument shows that if $\mathcal{B}$ does not abort, then, from the viewpoint of $\mathcal{A}$, the simulation provided by $\mathcal{B}$ is indistinguishable from a real attack scenario. In particular, (i) the view from $\mathcal{A}$ is independent of the value of c_{mode}, (ii) the public keys are uniformly distributed, and (iii) the signatures are correct. Therefore, $\mathcal{A}$ produces a valid forgery in time t with probability at least ϵ.
It remains to bound the probability that $\mathcal{B}$ does not abort. We argue as follows:

- Conditioned on the event $c_{\text{mode}} = b_{\text{type}} = 1$, Algorithm $\mathcal{B}$ aborts if $\mathcal{A}$ issued a signature query $m_\ell = w_\ell$. This happens with probability at most q_S/p.
- Conditioned on the event $c_{\text{mode}} = b_{\text{type}} = 2$, Algorithm $\mathcal{B}$ does not abort.

Since c_{mode} is independent of b_{type} we have that $\Pr[c_{\text{mode}} = b_{\text{type}}] = 1/2$. It now follows that $\mathcal{B}$ produces a valid forgery with probability at least $\epsilon/2 - q_S/p$, as required. □

Since in the full scheme a single message has many valid signatures, it is worth repeating that the full signature scheme is existentially unforgeable in the strong sense: the adversary cannot make any forgery, even on messages which are already signed.

3.3 Relation to Chameleon Hash Signatures

It is instructive to consider the relation between the full signature scheme above and a signature construction based on the Strong RSA assumption due to Gennaro, Halevi, and Rabin (GHR) [GHR99]. GHR signatures are pairs $(r, s^{1/H(m,r)})$ where H is a Chameleon hash [KR00], r is random in some range, and arithmetic is done modulo an RSA modulus N. Looking closely, one can see some parallels between the proof of security in Lemma 3.3 above and the proof of security in [GHR99]. There are three interesting points to make:

- The $m + yr$ component in our signature scheme provides us with the functionality of a Chameleon hash: given m, we can choose r so that $m + yr$ maps to some predefined value of our choice. This makes it possible to handle the chosen message attack. Embedding the hash $m + yr$ directly in the signature scheme results in a much more efficient construction than using an explicit Chameleon hash (which requires additional exponentiations). This is not known to be possible with Strong RSA signatures.

- One difficulty with GHR signatures is that given a solution $(6, s^{1/6})$ to the Strong RSA problem one can deduce another solution, e.g. $(3, s^{1/3})$. Thus, given a GHR signature on one message it possible to deduce a GHR signature on another message (see [GHR99, CN00] for

details). Gennaro et al. solve this problem by ensuring that $H(m, r)$ always maps to a prime; However, that makes it difficult to compute the hash (a different solution is given in [CS00]). This issue does not come up at all in our signature scheme above.

- We obtain short signatures since, unlike Strong RSA, the q-SDH assumption applies to groups with a short representation.

Thus, we see that Strong Diffie-Hellman leads to signatures that are simpler, more efficient, and shorter than their Strong RSA counterparts.

3.4 Limited Message Recovery

We now describe another useful property of the signature schemes whereby the total size of signed messages can be further reduced at the cost of increasing the verification time. The technique applies equally well to the fully secure signature scheme as to the weakly secure one.

A standard technique for shortening the total length of message/signature pairs is to encode a part of the message in the signature [MVV97]. Signatures based on trapdoor permutations support very efficient message recovery.

At the other end of the spectrum, a trivial signature compression mechanism that applies to any signature scheme is as follows: Rather than transmit a message/signature pair (M, σ), the sender transmits $(\hat{M}, \sigma)$ where $\hat{M}$ is the same as M except that the last t bits are truncated. In other words, $\hat{M}$ is t bits shorter than M. To verify $(\hat{M}, \sigma)$ the verifier tries all 2^t possible values for the truncated bits and accepts the signature if one of them verifies. To reconstruct the original signed message M, the verifier appends to $\hat{M}$ the t bits for which the signature verified.

This trivial method shows that the pair (M, σ) can be shortened by t-bits at the cost of increasing verification time by a factor of 2^t. For our signature scheme we obtain a better tradeoff: the pair (M, σ) can be shortened by t bits at the cost of increasing verification time by a factor of $2^{t/2}$ only. We refer to this property as limited message recovery.

Limited Message Recovery. Limited message recovery applies to both the full signature scheme and the weakly secure signature scheme of Lemma 3.2. For simplicity, we only show how limited message recovery applies to the full signature scheme. Assume messages are k-bit strings represented as integers in $\mathbb{Z}_p^*$. Let (g_1, g_2, u, v, z) be a public key in the full scheme—although for this application one might prefer to abbreviate the public key as (g_2, u, v) and let the verifier derive g_1 and z. Suppose we are given the signed message $(\hat{m}, \sigma, r)$ where $\hat{m}$ is a truncation of the last t bits of $m \in \mathbb{Z}_p^*$. Thus $m = \hat{m} \cdot 2^t + \delta$ for some integer $0 \le \delta < 2^t$. Our goal is to verify the signed message $(\hat{m}, \sigma, r)$ and to reconstruct the missing bits δ in time $2^{t/2}$. To do so, we first rewrite the verification equation $e(\sigma,\ u \cdot v^r \cdot g_2^m) = e(g_1, g_2)$ as

$$e(\sigma, g_2)^m = \frac{e(g_1, g_2)}{e(\sigma, u \cdot v^r)}$$

Substituting $m = \hat{m} \cdot 2^t + \delta$ we obtain

$$e(\sigma, g_2)^\delta = \frac{e(g_1, g_2)}{e(\sigma,\ u \cdot v^r \cdot g_2^{\hat{m}2^t})} \tag{2}$$

Now, we say that $(\hat{m}, \sigma, r)$ is valid if there exists an integer $\delta \in [0, 2^t)$ satisfying Equation (2). Finding such a δ takes time approximately $2^{t/2}$ using Pollard's Lambda method [MVV97, p.128] for computing discrete logarithms. Thus, we can verify the signature and recover the t missing message bits in time $2^{t/2}$, as required.

Ultra Short Weakly Secure Signatures. Obvious applications of limited message recovery are situations where bandwidth is extremely limited, such as when the signature is an authenticator that is to be typed-in by a human. The messages in such applications are typically chosen and signed by a central authority, so that adaptive chosen message attacks are typically not a concern. It is safe in those cases to use the weakly secure signature scheme of Lemma 3.2, and apply limited message recovery to further shrink the already compact signatures it produces. Specifically, using t-bit truncation as above we obtain a total signature overhead of $(160 - t)$ bits for common security parameters, at the cost of requiring $2^{t/2}$ arithmetic operations for signature verification. We emphasize that the security of this system does not rely on random oracles.

3.5 Arbitrary Message Signing

We can extend our signature schemes to sign arbitrary messages in $\{0,1\}^*$, as opposed to merely messages in $\mathbb{Z}_p^*$, by first hashing the message using a collision-resistant hash function $H : \{0,1\}^* \to \mathbb{Z}_p^*$ prior to both signing and verifying. A standard argument shows that if the scheme above is secure against existential forgery under a chosen message attack (in the strong sense) then so is the scheme with the hash. The result is a signature scheme for arbitrary messages in $\{0,1\}^*$. We note that there is no need for a full domain hash into $\mathbb{Z}_p^*$; a collision resistant hash function $H : \{0,1\}^* \to \{1, \ldots, 2^b\}$ for $2^b < p$ is sufficient for the security proof. This transformation applies to both the fully and the weakly secure signature schemes described above.

4 Shorter Signatures With Random Oracles

For completeness we show that the weakly secure signature scheme of Lemma 3.2 gives rise to very efficient and fully secure short signatures in the random oracle model. To do so, we show a general transformation from any existentially unforgeable signature scheme under a *weak* chosen message attack into an existentially unforgeable signature scheme under a *standard* chosen message attack (in the strong sense), in the random oracle model. This gives a very efficient short signature scheme based on q-SDH in the random oracle model. We analyze our construction using a method of Katz and Wang [KW03] which gives a very tight reduction to the security of the underlying signature. We note that a closely related system with a weaker security analysis was independently discovered by Zhang et al. [ZSNS04].

Let (*KeyGen*, *Sign*, *Verify*) be an existentially unforgeable signature under a *weak* chosen message attack. We assume that the scheme signs messages in some finite set Σ and that the private keys are in some set Π. We need two hash functions $H_1 : \Pi \times \{0,1\}^* \to \{0,1\}$ and $H_2 : \{0,1\} \times \{0,1\}^* \to \Sigma$ that will be viewed as random oracles in the security analysis. The hash-signature scheme is as follows:

Key generation: Same as *KeyGen*. The public key is PK; The secret key is $SK \in \Pi$.

Signing: Given a secret key SK, and given a message $M \in \{0,1\}^*$, compute $b \leftarrow H_1(SK, M) \in \{0,1\}$ and $m \leftarrow H_2(b, M) \in \Sigma$. Output the signature $(b, \mathit{Sign}(m))$. Note that signatures are one bit longer than in the underlying signature scheme.

Verification: Given a public key PK, a message $M \in \{0,1\}^*$, and a signature (b, σ), output `valid` if $\mathit{Verify}(PK, H_2(b, M), \sigma) = \texttt{valid}$.

Theorem 4.1 below proves security of the scheme. Note that the security reduction in Theorem 4.1 is tight, namely, an attacker on the hash-signature scheme with success probability ϵ is

converted to an attacker on the underlying signature with success probability approximately $\epsilon/2$. Proofs of signature schemes in the random oracle model are often far less tight.

Theorem 4.1. *Suppose (KeyGen, Sign, Verify) is (t', q'_S, ϵ')-existentially unforgeable under a weak chosen message attack. Then the corresponding hash-signature scheme is (t, q_S, q_H, ϵ)-secure against existential forgery under an adaptive chosen message attack, in the random oracle model, whenever $q_S + q_H < q'_S$, and for all all t and ϵ satisfying*

$$\epsilon \geq 2\epsilon'/(1 - \frac{q'_S}{|\Sigma|}) \approx 2\epsilon' \qquad \textit{and} \qquad t \leq t' - o(t')$$

Proof. Assume $\mathcal{A}$ is a forger that (t, q_S, q_H, ϵ)-breaks the hash-signature scheme (in the random oracle model). We construct an algorithm $\mathcal{B}$ that interacts with $\mathcal{A}$ and (t', q'_S, ϵ')-breaks the underlying signature scheme. Algorithm $\mathcal{B}$ works as follows:

Setup: Algorithm $\mathcal{B}$ picks q'_S random and independent messages $m_1, \ldots, m_{q'_s}$ in Σ and sends them to the challenger. The challenger responds with a public key PK and signatures $\sigma_1, \ldots, \sigma_{q'_s}$ on $m_1, \ldots, m_{q'_s}$. Algorithm $\mathcal{B}$ gives PK to Algorithm $\mathcal{A}$.

Hash queries: At any time Algorithm $\mathcal{A}$ can query the hash functions H_1 and H_2. It can query these functions q_H times each. Since $\mathcal{B}$ can maintain tables to ensure that repeated queries are answered consistently, we assume without loss of generality that $\mathcal{A}$ never queries on the same input twice.

To respond to a query for $H_1(K, M)$ Algorithm $\mathcal{B}$ first checks if $K = SK$ by attempting to sign a random message using K. If the signature is valid then $\mathcal{B}$ outputs that message/signature pair as an existential forgery and terminates. Otherwise, $\mathcal{B}$ picks a random bit $b \in \{0, 1\}$ and tells $\mathcal{A}$ that $H_1(K, M) = b$.

To respond to a query for $H_2(c, M)$ Algorithm $\mathcal{B}$ maintains a list of tuples (M_i, b_i, i) called the H-list, and a counter ℓ which is initially set to 0. The H-list is initially empty. When responding to a query for $H_2(c, M)$ we set things up so that we know the signature on either $H_2(0, M)$ or $H_2(1, M)$ but $\mathcal{A}$ will not know which one. More precisely, to respond to the query $H_2(c, M)$ Algorithm $\mathcal{B}$ does the following:

1. If M is not on the left hand side of any tuple in the H-list then pick a random bit $b \in \{0, 1\}$, set $\ell \leftarrow \ell + 1$, and add (M, b, ℓ) to the H-list.
2. Let (M, b, j) be the entry on the H-list corresponding to M. Then, if $b = c$ output $H_2(c, M) = m_j$ (for which we know that σ_j is a valid signature). Otherwise, pick a random message $m \in \Sigma$ and output $H_2(c, M) = m$. Note that $j \leq q_S + q_H < q'_S$ (since ℓ is always less than this value) so that m_j is well defined.

Signature queries: $\mathcal{A}$ can issue up to q_S signature queries. To respond to a signature query for $M \in \Sigma$ Algorithm $\mathcal{B}$ first runs the algorithm for responding to a hash query for $H_2(0, M)$ (hence the total number of H_2 queries is $q_S + q_H$). Let (M, b, j) be the entry on the H-list corresponding to M. Algorithm $\mathcal{B}$ responds with (b, σ_j) as the signature on M. This is a valid signature on M since $H_2(b, M) = m_j$ and σ_j is a valid signature on m_j. Note that this defines $H_1(SK, M) = b$ even though $\mathcal{B}$ does not know SK.

Output: Eventually, $\mathcal{A}$ returns a forgery, $(M_*, (b_*, \sigma_*))$, such that (b_*, σ_*) is a valid signature on M_* in the hash-signature scheme and $\mathcal{A}$ did not previously obtain (b_*, σ_*) from $\mathcal{B}$ in response

to a signature query on M_*. It follows that σ_* is a valid signature in the underlying signature scheme of the message $m_* = H_2(b_*, M_*)$. If $m_* \in \{m_1, \ldots, m_{q'_s}\}$ then $\mathcal{B}$ reports failure and aborts. Otherwise, it outputs (m_*, σ_*) as the existential forgery for the underlying signature scheme

Algorithm $\mathcal{B}$ simulates the random oracles and signature oracle perfectly for $\mathcal{A}$. Therefore $\mathcal{A}$ produces a valid forgery for the hash-signature scheme with probability at least ϵ. It remains to bound the probability that $m_* \in \{m_1, \ldots, m_{q'_s}\}$. Let (M_*, b, j) be the entry on the H-list corresponding to M_*. First, consider the case where $\mathcal{A}$ never issued a signature query for M_*. In this case the bit b is independent of $\mathcal{A}$'s view. Therefore, $\Pr[b_* = b] = 1/2$. Next, note that if $b_* = b$ then, by construction, $m_* = H_2(b_*, M_*) = m_j$ and therefore in this case $\mathcal{B}$ will fail. When $b_* \neq b$, by construction, $H_2(b_*, M_*)$ is chosen at random in Σ and therefore, in this case, $\mathcal{B}$ will fail with probability at most $q'_S/|\Sigma|$. Now, in the case where $\mathcal{A}$ did issue a signature query for M_*, we necessarily have $b_* \neq b$, otherwise $\mathcal{A}$'s forgery would be a replay of $\mathcal{B}$'s response. $\mathcal{B}$'s failure rate in this case is thus also at most $q'_S/|\Sigma|$. Thus, in all cases, it follows that $\mathcal{B}$ succeeds with probability at least

$$\Pr[\text{success}(\mathcal{B})] \geq \frac{\epsilon}{2} \cdot (1 - \frac{q'_S}{|\Sigma|}) \geq \epsilon'$$

as required. □

We note that in the proof above H_1 can be replaced with a Pseudo Random Function (PRF) and does not need to be modeled as a random oracle. However, modeling H_2 as a random oracles appears to be unavoidable.

Applying Theorem 4.1 to the weakly secure scheme of Lemma 3.2 gives an efficient short signature existentially unforgeable under a *standard* chosen message attack in the random oracle model assuming (q_S+q_H+1)-SDH. For a public key $(g_1,\ g_2,\ v = g_2^x,\ z)$ and a hash function $H : \{0,1\}^* \to \mathbb{Z}_p^*$ a signature on a message m is defined as the value $\sigma \leftarrow g_1^{1/(x+H(b,m))} \in \mathbb{G}_1$ concatenated with the bit $b \in \{0,1\}$. To verify the signature, check that $e(\sigma,\ v \cdot g_2^{H(b,m)}) = z = e(g_1, g_2)$. We see that signature length is essentially the same as in BLS signatures, but verification time is approximately half that of BLS. During verification, exponentiation is always base g_2 which enables a further speed-up by pre-computing certain powers of g_2.

Full Domain Hash. Another method for converting a signature scheme secure under a weak chosen message attack into a scheme secure under a standard chosen message attack is to simply apply *Sign* and *Verify* to $H(M)$ rather than M. In other words, we hash $M \in \{0,1\}^*$ using a full domain hash H prior to signing and verifying. Security in the random oracle model is shown using a similar argument to Coron's analysis [Cor00] of the Full Domain Hash [BR96]. However, the resulting reduction is not tight: an attacker on this hash-then-sign signature with success probability ϵ yields an attacker on the underlying signature with success probability approximately ϵ/q_S. We note, however, that these proofs are set in the random oracle model and therefore it is not clear whether the efficiency of the security reduction is relevant to actual security in the real world. Therefore, since this full domain hash signature scheme is slightly simpler that the system in Theorem 4.1 it might be preferable to use it rather than the system of Theorem 4.1. When we apply the full domain hash to the weakly secure scheme of Lemma 3.2, we obtain a secure signature under a standard chosen message attack assuming $(q_S + q_H + 1)$-SDH. A signature is one element, namely $\sigma \leftarrow g_1^{1/(x+H(m))} \in \mathbb{G}_1$. As before, signature verification is twice as fast as in BLS signatures. As mentioned above, a similar scheme was independently proposed by Zhang

et al. [ZSNS04]. We also note that, in the random oracle model, security of this full domain hash scheme can be proven under a slightly weaker complexity assumption than q-SDH, namely that the value c in the q-SDH assumption is pre-specified rather than chosen by the adversary. However, the resulting security reduction is far less efficient.

5 Generic Security of the q-SDH Assumption

To provide more confidence in the q-SDH assumption we prove a lower bound on the computational complexity of the q-SDH problem for generic groups in the sense of Shoup [Sho97].

In the generic group model, elements of $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ appear to be encoded as unique random strings, so that no property other than equality can be directly tested by the adversary. Five oracles are assumed to perform operations between group elements, such as computing the group action in each of the three groups $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, as well as the isomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$, and the bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. The opaque encoding of the elements of $\mathbb{G}_1$ is modeled as an injective function $\xi_1 : \mathbb{Z}_p \to \Xi_1$, where $\Xi_1 \subset \{0,1\}^*$, which maps all $a \in \mathbb{Z}_p$ to the string representation $\xi_1(g^a)$ of $g^a \in \mathbb{G}_1$. We similarly define $\xi_2 : \mathbb{Z}_p \to \Xi_2$ for $\mathbb{G}_2$ and $\xi_T : \mathbb{Z}_p \to \Xi_T$ for $\mathbb{G}_T$. The attacker $\mathcal{A}$ communicates with the oracles using the ξ-representations of the group elements only.

Theorem 5.1. *Let $\mathcal{A}$ be an algorithm that solves the q-SDH problem in the generic group model, making a total of at most q_G queries to the oracles computing the group action in $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$, the oracle computing the isomorphism ψ, and the oracle computing the bilinear pairing e. If $x \in \mathbb{Z}_p^*$ and ξ_1, ξ_2, ξ_T are chosen at random, then the probability ϵ that $\mathcal{A}(p, \xi_1(1), \xi_2(1), \xi_2(x), \ldots, \xi_2(x^q))$ outputs $(c, \xi_1(\frac{1}{x+c}))$ with $c \in \mathbb{Z}_p^*$, is bounded by*

$$\epsilon \leq \frac{(q_G + q + 2)^2 q}{p} = O\left(\frac{(q_G)^2 q + q^3}{p}\right)$$

Proof. Consider an algorithm $\mathcal{B}$ that plays the following game with $\mathcal{A}$.

$\mathcal{B}$ maintains three lists of pairs $L_1 = \{(F_{1,i}, \xi_{1,i}) : i = 0, \ldots, \tau_1 - 1\}$, $L_2 = \{(F_{2,i}, \xi_{2,i}) : i = 0, \ldots, \tau_2 - 1\}$, $L_T = \{(F_{T,i}, \xi_{T,i}) : i = 0, \ldots, \tau_T - 1\}$, such that, at step τ in the game, $\tau_1 + \tau_2 + \tau_T = \tau + q + 2$. The $F_{1,i}$ and $F_{2,i}$ are polynomials of degree $\leq q$ in $\mathbb{Z}_p[x]$, and the $F_{T,i}$ are polynomials of degree $\leq 2q$ in $\mathbb{Z}_p[x]$. The $\xi_{1,i}$, $\xi_{2,i}$, $\xi_{T,i}$ are strings in $\{0,1\}^*$. The lists are initialized at step $\tau = 0$ by taking $\tau_1 = 1$, $\tau_2 = q + 1$, $\tau_T = 0$, and posing $F_{1,0} = 1$, and $F_{2,i} = x^i$ for $i \in \{0, \ldots, q\}$. The corresponding $\xi_{1,0}$ and $\xi_{2,i}$ are set to arbitrary distinct strings in $\{0,1\}^*$.

We may assume that $\mathcal{A}$ only makes oracle queries on strings previously obtained form $\mathcal{B}$, since $\mathcal{B}$ can make them arbitrarily hard to guess. We note that $\mathcal{B}$ can determine the index i of any given string $\xi_{1,i}$ in L_1 (resp. $\xi_{2,i}$ in L_2, or $\xi_{T,i}$ in L_T), breaking ties between multiple matches arbitrarily.

$\mathcal{B}$ starts the game by providing $\mathcal{A}$ with the $q+2$ strings $\xi_{1,0}, \xi_{2,0}, \ldots, \xi_{2,q}$. Queries go as follows.

Group action: Given a multiply/divide selection bit and two operands $\xi_{1,i}$, $\xi_{1,j}$ with $0 \leq i, j < \tau_1$, we compute $F_{1,\tau_1} \leftarrow F_{1,i} \pm F_{1,j} \in \mathbb{Z}_p[x]$ depending on whether a multiplication or a division is requested. If $F_{1,\tau_1} = F_{1,l}$ for some $l < \tau_1$, we set $\xi_{1,\tau_1} \leftarrow \xi_{1,l}$; otherwise, we set ξ_{1,τ_1} to a string in $\{0,1\}^*$ distinct from $\xi_{1,0}, \ldots, \xi_{1,\tau_1-1}$. We add $(F_{1,\tau_1}, \xi_{1,\tau_1})$ to L_1 and give ξ_{1,τ_1} to $\mathcal{A}$, then increment τ_1 by one. Group action queries in $\mathbb{G}_2$ and $\mathbb{G}_T$ are treated similarly.

Isomorphism: Given a string $\xi_{2,i}$ with $0 \leq i < \tau_2$, we let $F_{1,\tau_1} \leftarrow F_{2,i} \in \mathbb{Z}_p[x]$. If $F_{1,\tau_1} = F_{1,l}$ for some $l < \tau_1$, we set $\xi_{1,\tau_1} \leftarrow \xi_{1,l}$; otherwise, we set ξ_{1,τ_1} to a string in $\{0,1\}^* \setminus \{\xi_{1,0}, \ldots, \xi_{1,\tau_1-1}\}$. We add $(F_{1,\tau_1}, \xi_{1,\tau_1})$ to L_1, give ξ_{1,τ_1} to $\mathcal{A}$, and increment τ_1 by one.

Pairing: Given two operands $\xi_{1,i}$ and $\xi_{2,j}$ with $0 \leq i < \tau_1$ and $0 \leq j < \tau_2$, we compute the product $F_{T,\tau_T} \leftarrow F_{1,i} \cdot F_{2,j} \in \mathbb{Z}_p[x]$. If $F_{T,\tau_T} = F_{T,l}$ for some $l < \tau_T$, we set $\xi_{T,\tau_T} \leftarrow \xi_{T,l}$; otherwise, we set ξ_{T,τ_T} to a string in $\{0,1\}^* \setminus \{\xi_{T,0}, \ldots, \xi_{T,\tau_T-1}\}$. We add $(F_{T,\tau_T}, \xi_{T,\tau_T})$ to L_T, give ξ_{T,τ_T} to $\mathcal{A}$, and increment τ_T by one.

$\mathcal{A}$ terminates and returns a pair $(c, \xi_{1,\ell})$ where $0 \leq \ell < \tau_1$. Let $F_{1,\ell}$ be the corresponding polynomial in the list L_1. In order to exhibit the correctness of $\mathcal{A}$'s answer within the simulation framework, $\mathcal{B}$ computes the polynomial $F_{T,\star} = F_{1,\ell} \cdot (F_{2,1} + [c]F_{2,0}) = F_{1,\ell} \cdot (x + c)$. Notice that if $\mathcal{A}$'s answer is correct then necessarily

$$F_{T,\star}(x) - 1 = 0 \tag{3}$$

Indeed, this equality corresponds to the DDH relation "$e(A, g_2^x g_2^c) = e(g_1, g_2)$" where A denotes the element of $\mathbb{G}_1$ represented by $\xi_{1,\ell}$. Observe that since the constant monomial "1" has degree 0 and $F_{T,\star} = F_{1,\ell} \cdot (x+c)$ where $(x+c)$ has degree 1, the above relation (3) cannot be satisfied identically in $\mathbb{Z}_p[x]$ unless $F_{1,\ell}$ has degree $\geq p-2$. We know that the degree of $F_{1,\ell}$ is at most q, therefore we deduce that there exists a value of x for which Equation (3) does not hold. Thus, since it is a non-trivial polynomial equation of degree $\leq q+1$, it admits at most $q+1$ roots in $\mathbb{Z}_p$.

At this point $\mathcal{B}$ chooses a random $x^* \in \mathbb{Z}_p$. The simulation provided by $\mathcal{B}$ is perfect unless the instantiation $x \leftarrow x^*$ creates an equality relation between the simulated group elements that was not revealed to $\mathcal{A}$, a category in which relation (3) belongs, as we just shown. Thus, the success probability of $\mathcal{A}$ is bounded by the probability that any of the following holds:

1. $F_{1,i}(x^*) - F_{1,j}(x^*) = 0$ for some i, j such that $F_{1,i} \neq F_{1,j}$,
2. $F_{2,i}(x^*) - F_{2,j}(x^*) = 0$ for some i, j such that $F_{2,i} \neq F_{2,j}$,
3. $F_{T,i}(x^*) - F_{T,j}(x^*) = 0$ for some i, j such that $F_{T,i} \neq F_{T,j}$,
4. $F_{1,\ell}(x^*) \cdot (x^* + c) - 1 = 0$.

Since $F_{1,i} - F_{1,j}$ for fixed i and j is a polynomial of degree at most q, it vanishes at a random $x^* \in \mathbb{Z}_p$ with probability at most q/p. Similarly, for fixed i and j, the second case occurs with probability $\leq q/p$, the third with probability $\leq 2q/p$ (since $F_{T,i} - F_{T,j}$ has degree at most $2q$), and the fourth with probability $\leq (q+1)/p$. By summing over all valid pairs (i, j) in each case, we find that $\mathcal{A}$ wins the game with probability $\epsilon \leq \binom{\tau_1}{2}\frac{q}{p} + \binom{\tau_2}{2}\frac{q}{p} + \binom{\tau_T}{2}\frac{2q}{p} + \frac{q+1}{p}$. Since $\tau_1 + \tau_2 + \tau_T \leq q_G + q + 2$, the required bound follows: $\epsilon \leq (q_G + q + 2)^2(q/p) = O((q_G)^2(q/p) + q^3/p)$. □

Corollary 5.2. *Any adversary that solves the q-SDH problem with constant probability $\epsilon > 0$ in generic groups of order p such that $q < o(\sqrt[3]{p})$ requires $\Omega(\sqrt{\epsilon p/q})$ generic group operations.*

6 Conclusions

We presented a number of short signature schemes based on the q-SDH assumption. Our main result is a short signature which is fully secure without using the random oracle model. The signature is as short as DSA signatures, but is provably secure in the standard model. We also showed that the scheme supports limited message recovery, for even greater compactness.

These constructions are possible thanks to properties of the q-SDH assumption. The assumption can be viewed as a discrete logarithm analogue of the Strong RSA assumption. We believe the q-SDH assumption is a useful tool for constructing cryptographic systems and we expect to see many other schemes based on it. For example, we mention a new group signature scheme of Boneh et al. [BBS04].

Acknowledgments

We thank Mihir Bellare and Nigel Smart for their helpful comments on this paper.

References

[ADR02] Jee Hea An, Yevgeniy Dodis, and Tal Rabin. On the security of joint signature and encryption. In *Proceedings of Eurocrypt 2002*, volume 2332 of *LNCS*. Springer-Verlag, 2002.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *Proceedings of Crypto '04*, 2004.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In *Proceedings of Asiacrypt 2001*, volume 2248 of *LNCS*, pages 514–32. Springer-Verlag, 2001.

[BR93] Mihir Bellare and Phil Rogaway. Random oracle are practical: A paradigm for designing efficient protocols. In *Proceedings of the First ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[BR96] Mihir Bellare and Phil Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In Ueli Maurer, editor, *Proceedings of Eurocrypt '96*, volume 1070 of *LNCS*, pages 399–416. Springer-Verlag, 1996.

[CDF03] Nicolas Courtois, Magnus Daum, and Patrick Felke. On the security of HFE, HFEv- and Quartz. In *Proceedings of PKC 2003*, volume 2567 of *LNCS*, pages 337–50. Springer-Verlag, 2003.

[CHK04] Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. In *Advances in Cryptology—EUROCRYPT '04*, volume 3027 of *LNCS*, pages 207–222. Springer-Verlag, 2004.

[CN00] Jean-Sébastien Coron and David Naccache. Security analysis of the Gennaro-Halevi-Rabin signature scheme. In *Proceedings of Eurocrypt 2000*, pages 91–101, 2000.

[Cor00] Jean-Sébastien Coron. On the exact security of full domain hash. In *Proceedings of Crypto 2000*, volume 1880 of *LNCS*, pages 229–35. Springer-Verlag, 2000.

[CS00] Ronald Cramer and Victor Shoup. Signature schemes based on the strong RSA assumption. *ACM TISSEC*, 3(3):161–185, 2000. Extended abstract in Proc. 6th ACM CCS, 1999.

[GHR99] Rosario Gennaro, Shai Halevi, and Tal Rabin. Secure hash-and-sign signatures without the random oracle. In *Proceedings of Eurocrypt 1999*, LNCS, pages 123–139. Springer-Verlag, 1999.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ron Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2):281–308, 1988.

[JN01] Antoine Joux and Kim Nguyen. Separating decision Diffie-Hellman from Diffie-Hellman in cryptographic groups. Cryptology ePrint Archive, Report 2001/003, 2001. `http://eprint.iacr.org/2001/003/`.

[KR00] Hugo Krawczyk and Tal Rabin. Chameleon signatures. In *Proceedings of NDSS 2000*. Internet Society, 2000. `http://eprint.iacr.org/1998/010/`.

[KW03] Jonathan Katz and Nan Wang. Efficiency improvements for signature schemes with tight security reductions. In *Proceedings of ACM CCS*, 2003.

[MSK02] Shigeo Mitsunari, Ryuichi Sakai, and Masao Kasahara. A new traitor tracing. *IEICE Trans. Fundamentals*, E85-A(2):481–484, 2002.

[MVV97] Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[NS00] David Naccache and Jacques Stern. Signing on a postcard. In *Proceedings of Financial Cryptography 2000*, 2000.

[PCG01] Jacques Patarin, Nicolas Courtois, and Louis Goubin. QUARTZ, 128-bit long digital signatures. In *Proceedings of RSA 2001*, volume 2020 of *LNCS*, pages 282–97. Springer-Verlag, 2001.

[PV00] Leon Pintsov and Scott Vanstone. Postal revenue collection in the digital age. In *Proceedings of Financial Cryptography 2000*, 2000.

[Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *Proceeings 40 IEEE Symp. on Foundations of Computer Science*, 1999.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *Proceedings of Eurocrypt 1997*. Springer-Verlag, 1997.

[TSNZ03] Vu Dong Tô, Reihaneh Safavi-Naini, and Fangguo Zhang. New traitor tracing schemes using bilinear map. In *Proceedings of 2003 DRM Workshop*, 2003.

[ZSNS04] Fangguo Zhang, Reihaneh Safavi-Naini, and Willy Susilo. An efficient signature scheme from bilinear pairings and its applications. In *Proceedings of PKC 2004*, 2004.

Public Key Encryption with keyword Search

Dan Boneh
Stanford University

Giovanni Di Crescenzo
Telcordia

Rafail Ostrovsky
UCLA

Giuseppe Persiano
Università di Salerno

Abstract

We study the problem of searching on data that is encrypted using a public key system. Consider user Bob who sends email to user Alice encrypted under Alice's public key. An email gateway wants to test whether the email contains the keyword "urgent" so that it could route the email accordingly. Alice, on the other hand does not wish to give the gateway the ability to decrypt all her messages. We define and construct a mechanism that enables Alice to provide a key to the gateway that enables the gateway to test whether the word "urgent" is a keyword in the email without learning anything else about the email. We refer to this mechanism as *Public Key Encryption with keyword Search*. As another example, consider a mail server that stores various messages publicly encrypted for Alice by others. Using our mechanism Alice can send the mail server a key that will enable the server to identify all messages containing some specific keyword, but learn nothing else. We define the concept of public key encryption with keyword search and give several constructions.

1 Introduction

Suppose user Alice wishes to read her email on a number of devices: laptop, desktop, pager, etc. Alice's mail gateway is supposed to route email to the appropriate device based on the keywords in the email. For example, when Bob sends email with the keyword "urgent" the mail is routed to Alice's pager. When Bob sends email with the keyword "lunch" the mail is routed to Alice's desktop for reading later. One expects each email to contain a small number of keywords. For example, all words on the subject line as well as the sender's email address could be used as keywords. The mobile people project [24] provides this email processing capability.

Now, suppose Bob sends encrypted email to Alice using Alice's public key. Both the contents of the email and the keywords are encrypted. In this case the mail gateway cannot see the keywords and hence cannot make routing decisions. As a result, the mobile people project is unable to process secure email without violating user privacy. Our goal is to enable Alice to give the gateway the ability to test whether "urgent" is a keyword in the email, but the gateway should learn nothing else about the email. More generally, Alice should be able to specify a few keywords that the mail gateway can search for, but learn nothing else about incoming mail. We give precise definitions in section 2.

To do so, Bob encrypts his email using a standard public key system. He then appends to the resulting ciphertext a *Public-Key Encryption with keyword Search* (PEKS) of each keyword. To send a message M with keywords $W_1, \ldots, W_m$ Bob sends

$$E_{A_{pub}}(M) \;\|\; \mathsf{PEKS}(A_{pub}, W_1) \;\|\; \cdots \;\|\; \mathsf{PEKS}(A_{pub}, W_m)$$

Where A_{pub} is Alice's public key. The point of this form of encryption is that Alice can give the gateway a certain trapdoor T_W that enables the gateway to test whether one of the keywords associated with the message is equal to the word W of Alice's choice. Given $\mathsf{PEKS}(A_{pub}, W')$ and T_W the gateway can test whether $W = W'$. If $W \neq W'$ the gateway learns nothing more about W'. Note that Alice and Bob do not communicate in this entire process. Bob generates the searchable encryption for W' just given Alice's public key.

In some cases, it is instructive to view the email gateway as an IMAP or POP email server. The server stores many emails and each email contains a small number of keywords. As before, all these emails are created by various people sending mail to Alice encrypted using her public key. We want to enable Alice to ask queries of the form: do any of the messages on the server contain the keyword "urgent"? Alice would do this by giving the server a trapdoor T_W, thus enabling the server to retrieve emails containing the keyword W. The server learns nothing else about the emails.

Related work. A related issue deals with privacy of database data. There are two different scenarios: *public* databases and *private* databases, and the solutions for each are different.
Private databases: In this settings a user wishes to upload its private data to a remote database and wishes to keep the data private from the remote database administrator. Later, the user must be able to retrieve from the remote database all records that contain a particular keyword. Solutions to this problem were presented in the early 1990's by Ostrovsky [26] and Ostrovsky and Goldreich [17] and more recently by Song at al. [28]. The solution of Song. at al [28] requires very little communication between the user and the database (proportional to the security parameter) and only one round of interaction. The database performs work that is linear in its size per query. The solution of [26, 17] requires poly-logarithmic rounds (in the size of the database) between the user and the database, but allows the database to do only poly-logarithmic work per query. An additional privacy requirement that might be appealing in some scenarios is to hide from the database administrator any information regarding the *access pattern*, i.e. if some item was retrieved more then once, some item was not retrieved at all, etc. The work of [26, 17] achieves this property as well, with the same poly-logarithmic cost[1] per query both for the database-user interaction and the actual database work. We stress that both the constructions of [26, 17] and the more recent work of [10, 28, 16] apply only to the private-key setting for users who own their data and wish to upload it to a third-party database that they do not trust.
Public Databases Here the database data is public (such as stock quotes) but the user is unaware of it and wishes to retrieve some data-item or search for some data-item, without revealing to the database administrator which item it is. The naive solution is that the user can download the entire database. Public Information Retrieval (PIR) protocols allow user to retrieve data from a public database with far smaller communication then just downloading the entire database. PIR was first shown to be possible only in the setting where there are many copies of the same database and none of the copies can talk to each other [5]. PIR was shown to be possible for a single database by Kushilevitz and Ostrovsky [22] (using homomorphic encryption scheme of [19]). The communication complexity of [22] solution (i.e. the number of bits transmitted between the user

[1]The poly-logarithmic construction of [26, 17] requires large constants, which makes it impractical; however their basic $O(\sqrt{n})$ solution was recently shown to be applicable for some practical applications [10].

and the database) is $O(n^\epsilon)$, where n is the size of the database and $\epsilon > 0$. This was reduced to poly-logarithmic overhead by Cachin, Micali, and Stadler [4]. As pointed out in [22], the model of PIR can be extended to one-out-of-n Oblivious Transfer and keyword searching on public data, and received a lot of additional attention in the literature (see, for example, [22, 8, 20, 9, 23, 25, 27]. We stress though that in all these settings the database is public, and the user is trying to retrieve or find certain items without revealing to the database administrator what it is searching for. In the setting of a single public database, it can be shown that the database must always perform work which is at least linear in the size of the database.

Our problem does not fit either of the two models mentioned above. Unlike the private-key setting, data collected by the mail-server is from third parties, and can not be "organized" by the user in any convenient way. Unlike the publicly available database, the data is not public, and hence the PIR solutions do not apply.

We point out that in practical applications, due to the computation cost of public key encryption, our constructions are applicable to searching on a small number of keywords rather than an entire file. Recently, Waters et al. [30] showed that public key encryption with keyword search can be used to build an encrypted and searchable audit log. Other methods for searching on encrypted data are described in [16, 12].

2 Public key encryption with searching: definitions

Throughout the paper we use the term *negligible function* to refer to a function $f : \mathbb{R} \to [0, 1]$ where $f(s) < 1/g(s)$ for any polynomial g and sufficiently large s.

We start by precisely defining what is a secure Public Key Encryption with keyword Search (PEKS) scheme. Here "public-key" refers to the fact that ciphertexts are created by various people using Alice's public key. Suppose user Bob is about to send an encrypted email to Alice with keywords $W_1, \ldots, W_k$ (e.g., words in the subject line and the sender's address could be used as keywords, so that k is relatively small). Bob sends the following message:

$$\left[E_{A_{pub}}[msg],\ \mathsf{PEKS}(A_{pub}, W_1), \ldots, \mathsf{PEKS}(A_{pub}, W_k)\right] \tag{1}$$

where A_{pub} is Alice's public key, msg is the email body, and PEKS is an algorithm with properties discussed below. The PEKS values do not reveal any information about the message, but enable searching for specific keywords. For the rest of the paper, we use as our sample application a mail server that stores all incoming email.

Our goal is to enable Alice to send a short secret key T_W to the mail server that will enable the server to locate all messages containing the keyword W, but learn nothing else. Alice produces this trapdoor T_W using her private key. The server simply sends the relevant emails back to Alice. We call such a system *non-interactive public key encryption with keyword search*, or as a shorthand "searchable public-key encryption".

Definition 2.1. A non-interactive public key encryption with keyword search (we sometimes abbreviate it as "searchable encryption") scheme consists of the following polynomial time randomized algorithms:

1. $\mathsf{KeyGen}(s)$: Takes a security parameter, s, and generates a public/private key pair A_{pub}, A_{priv}.
2. $\mathsf{PEKS}(A_{pub}, W)$: for a public key A_{pub} and a word W, produces a searchable encryption of W.
3. $\mathsf{Trapdoor}(A_{priv}, W)$: given Alice's private key and a word W produces a trapdoor T_W.
4. $\mathsf{Test}(A_{pub}, S, T_W)$: given Alice's public key, a searchable encryption $S = \mathsf{PEKS}(A_{pub}, W')$, and a trapdoor $T_W = \mathsf{Trapdoor}(A_{priv}, W)$, outputs 'yes' if $W = W'$ and 'no' otherwise.

Alice runs the KeyGen algorithm to generate her public/private key pair. She uses Trapdoor to generate trapdoors T_W for any keywords W that she wants the mail server or mail gateway to search for. The mail server uses the given trapdoors as input to the Test() algorithm to determine whether a given email contains one of the keywords W specified by Alice.

Next, we define security for a PEKS in the sense of semantic-security. We need to ensure that an $\mathsf{PEKS}(A_{pub}, W)$ does not reveal any information about W unless T_W is available. We define security against an active attacker who is able to obtain trapdoors T_W for any W of his choice. Even under such attack the attacker should not be able to distinguish an encryption of a keyword W_0 from an encryption of a keyword W_1 for which he did not obtain the trapdoor. Formally, we define security against an active attacker $\mathcal{A}$ using the following game between a challenger and the attacker (the security parameter s is given to both players as input).

PEKS Security game:

1. The challenger runs the $\mathsf{KeyGen}(s)$ algorithm to generate A_{pub} and A_{priv}. It gives A_{pub} to the attacker.
2. The attacker can adaptively ask the challenger for the trapdoor T_W for any keyword $W \in \{0,1\}^*$ of his choice.
3. At some point, the attacker $\mathcal{A}$ sends the challenger two words W_0, W_1 on which it wishes to be challenged. The only restriction is that the attacker did not previously ask for the trapdoors T_{W_0} or T_{W_1}. The challenger picks a random $b \in \{0,1\}$ and gives the attacker $C = \mathsf{PEKS}(A_{pub}, W_b)$. We refer to C as the challenge PEKS.
4. The attacker can continue to ask for trapdoors T_W for any keyword W of his choice as long as $W \neq W_0, W_1$.
5. Eventually, the attacker $\mathcal{A}$ outputs $b' \in \{0,1\}$ and wins the game if $b = b'$.

In other words, the attacker wins the game if he can correctly guess whether he was given the PEKS for W_0 or W_1. We define $\mathcal{A}$'s advantage in breaking the PEKS as

$$\mathrm{Adv}_{\mathcal{A}}(s) = |\Pr[b = b'] - \frac{1}{2}|$$

Definition 2.2. We say that a PEKS is semantically secure against an adaptive chosen keyword attack if for any polynomial time attacker $\mathcal{A}$ we have that $\mathrm{Adv}_{\mathcal{A}}(s)$ is a negligible function.

Chosen Ciphertext Security. We note that Definition 2.2 ensures that the construction given in Eq. (1) is semantically secure whenever the public key encryption system $E_{A_{pub}}$ is semantically secure. However, as is, the construction is not chosen ciphertext secure. Indeed, a chosen ciphertext attacker can break semantic security by reordering the keywords in Eq. (1) and submitting the resulting ciphertext for decryption. A standard technique can make this construction chosen ciphertext secure using the methods of [7]. We defer this to the full version of the paper.

2.1 PEKS implies Identity Based Encryption

Public key encryption with keyword search is related to Identity Based Encryption (IBE) [29, 2]. Constructing a secure PEKS appears to be a harder problem than constructing an IBE. Indeed, the following lemma shows that PEKS implies Identity Based Encryption. The converse is probably false. Security notions for IBE, and in particular chosen ciphertext secure IBE (IND-ID-CCA), are defined in [2].

Lemma 2.3. *A non-interactive searchable encryption scheme (*PEKS*) that is semantically secure against an adaptive chosen keyword attack gives rise to a chosen ciphertext secure IBE system (IND-ID-CCA).*

Proof sketch: Given a PEKS (KeyGen, PEKS, Trapdoor, Test) the IBE system is as follows:

1. Setup: Run the PEKS KeyGen algorithm to generate A_{pub}/A_{priv}. The IBE system parameters are A_{pub}. The master-key is A_{priv}.
2. KeyGen: The IBE private key associated with a public key $X \in \{0,1\}^*$ is

$$d_X = [\mathsf{Trapdoor}(A_{priv}, X\|0),\ \ \mathsf{Trapdoor}(A_{priv}, X\|1)]\,,$$

   where $\|$ denotes concatenation.
3. Encrypt: Encrypt a bit $b \in \{0,1\}$ using a public key $X \in \{0,1\}^*$ as: $CT = \mathsf{PEKS}(A_{pub}, X\|b)$.
4. Decrypt: To decrypt $CT = \mathsf{PEKS}(A_{pub}, X\|b)$ using the private key $d_X = (d_0, d_1)$. Output '0' if $\mathsf{Test}(A_{pub}, CT, d_0) =$ 'yes' and output '1' if $\mathsf{Test}(A_{pub}, CT, d_1) =$ 'yes'

One can show that the resulting system is IND-ID-CCA assuming the PEKS is semantically secure against an adaptive chosen message attack. □

This shows that building non-interactive public-key searchable encryption is at least as hard as building an IBE system. One might be tempted to prove the converse (i.e., IBE implies PEKS) by defining

$$\mathsf{PEKS}(A_{pub}, W) = E_W[0^k] \tag{2}$$

i.e. encrypt a string of k zeros with the IBE public key $W \in \{0,1\}^*$. The Test algorithm attempts to decrypt $E_W[0]$ and checks that the resulting plaintext is 0^k. Unfortunately, this does not necessarily give a secure searchable encryption scheme. The problem is that the ciphertext CT could expose the public key (W) used to create CT. Generally, an encryption scheme need not hide the public key that was used to create a given ciphertext. But this property is essential for the PEKS construction given in (2). We note that public key privacy was previously studied by Bellare et al. [1]. The construction in (2) requires an IBE system that is public-key private.

Generally, it appears that constructing a searchable public-key encryption is a harder problem than constructing an IBE scheme. Nevertheless, our first PEKS construction is based on a recent construction for an IBE system. We are able to prove security by exploiting extra properties of this system.

3 Constructions

We give two constructions for public-key searchable encryption: (1) an efficient system based on a variant of the Decision Diffie-Hellman assumption (assuming a random oracle) and (2) a limited system based on general trapdoor permutations (without assuming the random oracle), but less efficient.

3.1 Construction using bilinear maps

Our first construction is based on a variant of the Computational Diffie-Hellman problem. Boneh and Franklin [2] recently used bilinear maps on elliptic curves to build an efficient IBE system. Abstractly, they use two groups G_1, G_2 of prime order p and a bilinear map $e : G_1 \times G_1 \to G_2$ between them. The map satisfies the following properties:

1. Computable: given $g, h \in G_1$ there is a polynomial time algorithms to compute $e(g, h) \in G_2$.
2. Bilinear: for any integers $x, y \in [1, p]$ we have $e(g^x, g^y) = e(g, g)^{xy}$
3. Non-degenerate: if g is a generator of G_1 then $e(g, g)$ is a generator of G_2.

The size of G_1, G_2 is determined by the security parameter.

We build a non-interactive searchable encryption scheme from such a bilinear map. The construction is based on [2]. We will need hash functions $H_1 : \{0,1\}^* \to G_1$ and $H_2 : G_2 \to \{0,1\}^{\log p}$. Our PEKS works as follows:

- KeyGen: The input security parameter determines the size, p, of the groups G_1 and G_2. The algorithm picks a random $\alpha \in \mathbb{Z}_p^*$ and a generator g of G_1. It outputs $A_{pub} = [g, h = g^\alpha]$ and $A_{priv} = \alpha$.
- PEKS(A_{pub}, W): First compute $t = e(H_1(W), h^r) \in G_2$ for a random $r \in \mathbb{Z}_p^*$. Output PEKS$(A_{pub}, W) = [g^r,\ H_2(t)]$.
- Trapdoor(A_{priv}, W): output $T_W = H_1(W)^\alpha \in G_1$.
- Test(A_{pub}, S, T_W): let $S = [A, B]$. Test if $H_2(e(T_W, A)) = B$. If so, output 'yes'; if not, output 'no'.

We prove that this system is a non-interactive searchable encryption scheme semantically secure against a chosen keyword attack in the random oracle model. The proof of security relies on the difficulty of the Bilinear Diffie-Hellman problem (BDH) [2, 21].

Bilinear Diffie-Hellman Problem (BDH): Fix a generator g of G_1. The BDH problem is as follows: given $g, g^a, g^b, g^c \in G_1$ as input, compute $e(g, g)^{abc} \in G_2$. We say that BDH is intractable if all polynomial time algorithms have a negligible advantage in solving BDH.

We note that the Boneh-Franklin IBE system [2] relies on the same intractability assumption for security. The security of our PEKS is proved in the following theorem. The proof is set in the random oracle model. Indeed, it is currently an open problem to build a secure IBE, and hence a PEKS, without the random oracle model.

Theorem 3.1. *The non-interactive searchable encryption scheme (PEKS) above is semantically secure against a chosen keyword attack in the random oracle model assuming BDH is intractable.*

Proof : Suppose $\mathcal{A}$ is an attack algorithm that has advantage ϵ in breaking the PEKS. Suppose $\mathcal{A}$ makes at most q_{H_2} hash function queries to H_2 and at most q_T trapdoor queries (we assume q_T and q_{H_2} are positive). We construct an algorithm $\mathcal{B}$ that solves the BDH problem with probability at least $\epsilon' = \epsilon/(e q_T q_{H_2})$, where e is the base of the natural logarithm. Algorithm $\mathcal{B}$'s running time is approximately the same as $\mathcal{A}$'s. Hence, if the BDH assumption holds in G_1 then ϵ' is a negligible function and consequently ϵ must be a negligible function in the security parameter.

Let g be a generator of G_1. Algorithm $\mathcal{B}$ is given $g, u_1 = g^\alpha, u_2 = g^\beta, u_3 = g^\gamma \in G_1$. Its goal is to output $v = e(g, g)^{\alpha\beta\gamma} \in G_2$. Algorithm $\mathcal{B}$ simulates the challenger and interacts with forger $\mathcal{A}$ as follows:

KeyGen. Algorithm $\mathcal{B}$ starts by giving $\mathcal{A}$ the public key $A_{pub} = [g, u_1]$.

H_1, H_2**-queries.** At any time algorithm $\mathcal{A}$ can query the random oracles H_1 or H_2. To respond to H_1 queries algorithm $\mathcal{B}$ maintains a list of tuples $\langle W_j, h_j, a_j, c_j \rangle$ called the H_1-list. The list is initially empty. When $\mathcal{A}$ queries the random oracle H_1 at a point $W_i \in \{0,1\}^*$, algorithm $\mathcal{B}$ responds as follows:

1. If the query W_i already appears on the H_1-list in a tuple $\langle W_i, h_i, a_i, c_i \rangle$ then algorithm $\mathcal{B}$ responds with $H_1(W_i) = h_i \in G_1$.
2. Otherwise, $\mathcal{B}$ generates a random coin $c_i \in \{0,1\}$ so that $\Pr[c_i = 0] = 1/(q_T + 1)$.
3. Algorithm $\mathcal{B}$ picks a random $a_i \in \mathbb{Z}_p$.
   If $c_i = 0$, $\mathcal{B}$ computes $h_i \leftarrow u_2 \cdot g^{a_i} \in G_1$.
   If $c_i = 1$, $\mathcal{B}$ computes $h_i \leftarrow g^{a_i} \in G_1$.
4. Algorithm $\mathcal{B}$ adds the tuple $\langle W_i, h_i, a_i, c_i \rangle$ to the H_1-list and responds to $\mathcal{A}$ by setting $H_1(W_i) = h_i$. Note that either way h_i is uniform in G_1 and is independent of $\mathcal{A}$'s current view as required.

Similarly, at any time $\mathcal{A}$ can issue a query to H_2. Algorithm $\mathcal{B}$ responds to a query for $H_2(t)$ by picking a new random value $V \in \{0,1\}^{\log p}$ for each new t and setting $H_2(t) = V$. In addition, $\mathcal{B}$ keeps track of all H_2 queries by adding the pair (t, V) to an H_2-list. The H_2-list is initially empty.

Trapdoor queries. When $\mathcal{A}$ issues a query for the trapdoor corresponding to the word W_i algorithm $\mathcal{B}$ responds as follows:

1. Algorithm $\mathcal{B}$ runs the above algorithm for responding to H_1-queries to obtain an $h_i \in G_1$ such that $H_1(W_i) = h_i$. Let $\langle W_i, h_i, a_i, c_i \rangle$ be the corresponding tuple on the H_1-list. If $c_i = 0$ then $\mathcal{B}$ reports failure and terminates.
2. Otherwise, we know $c_i = 1$ and hence $h_i = g^{a_i} \in G_1$. Define $T_i = u_1^{a_i}$. Observe that $T_i = H(W_i)^\alpha$ and therefore T_i is the correct trapdoor for the keyword W_i under the public key $A_{pub} = [g, u_1]$. Algorithm $\mathcal{B}$ gives T_i to algorithm $\mathcal{A}$.

Challenge. Eventually algorithm $\mathcal{A}$ produces a pair of keywords W_0 and W_1 that it wishes to be challenged on. Algorithm $\mathcal{B}$ generates the challenge PEKS as follows:

1. Algorithm $\mathcal{B}$ runs the above algorithm for responding to H_1-queries twice to obtain a $h_0, h_1 \in G_1$ such that $H_1(W_0) = h_0$ and $H_1(W_1) = h_1$. For $i = 0, 1$ let $\langle W_i, h_i, a_i, c_i \rangle$ be the corresponding tuples on the H_1-list. If both $c_0 = 1$ and $c_1 = 1$ then $\mathcal{B}$ reports failure and terminates.
2. We know that at least one of c_0, c_1 is equal to 0. Algorithm $\mathcal{B}$ randomly picks a $b \in \{0,1\}$ such that $c_b = 0$ (if only one c_b is equal to 0 then no randomness is needed since there is only one choice).
3. Algorithm $\mathcal{B}$ responds with the challenge PEKS $C = [u_3, J]$ for a random $J \in \{0,1\}^{\log p}$.

Note that this challenge implicitly defines $H_2(e(H_1(W_b), u_1^\gamma)) = J$. In other words,

$$J = H_2(e(H_1(W_b), u_1^\gamma)) = H_2(e(u_2 g^{a_b}, g^{\alpha\gamma})) = H_2(e(g,g)^{\alpha\gamma(\beta + a_b)})$$

With this definition, C is a valid PEKS for W_b as required.

More trapdoor queries. $\mathcal{A}$ can continue to issue trapdoor queries for keywords W_i where the only restriction is that $W_i \neq W_0, W_1$. Algorithm $\mathcal{B}$ responds to these queries as before.

Output. Eventually, $\mathcal{A}$ outputs its guess $b' \in \{0,1\}$ indicating whether the challenge C is the result of $\mathsf{PEKS}(A_{pub}, W_0)$ or $\mathsf{PEKS}(A_{pub}, W_1)$. At this point, algorithm $\mathcal{B}$ picks a random pair (t, V) from the H_2-list and outputs $t/e(u_1, u_3)^{a_b}$ as its guess for $e(g,g)^{\alpha\beta\gamma}$, where a_b is the value used in the Challenge step. The reason this works is that, as we will show, $\mathcal{A}$ must have issued a query for either $H_2(e(H_1(W_0), u_1^{\gamma}))$ or $H_2(e(H_1(W_1), u_1^{\gamma}))$. Therefore, with probability 1/2 the H_2-list contains a pair whose left hand side is $t = e(H_1(W_b), u_1^{\gamma}) = e(g,g)^{\alpha\gamma(\beta+a_b)}$. If $\mathcal{B}$ picks this pair (t, V) from the H_2-list then $t/e(u_1, u_3)^{a_b} = e(g,g)^{\alpha\beta\gamma}$ as required.

This completes the description of algorithm $\mathcal{B}$. It remains to show that $\mathcal{B}$ correctly outputs $e(g,g)^{\alpha\beta\gamma}$ with probability at least ϵ'. To do so, we first analyze the probability that $\mathcal{B}$ does not abort during the simulation. We define two events:

$\mathcal{E}_1$**:** $\mathcal{B}$ does not abort as a result of any of $\mathcal{A}$'s trapdoor queries.
$\mathcal{E}_2$**:** $\mathcal{B}$ does not abort during the challenge phase.

We first argue as in [6] that both events $\mathcal{E}_1$ and $\mathcal{E}_2$ occur with sufficiently high probability.

Claim 1: The probability that algorithm $\mathcal{B}$ does not abort as a result of $\mathcal{A}$'s trapdoor queries is at least $1/e$. Hence, $\Pr[\mathcal{E}_1] \geq 1/e$.

Proof. Without loss of generality we assume that $\mathcal{A}$ does not ask for the trapdoor of the same keyword twice. The probability that a trapdoor query causes $\mathcal{B}$ to abort is $1/(q_T + 1)$. To see this, let W_i be $\mathcal{A}$'s i'th trapdoor query and let $\langle W_i, h_i, a_i, c_i \rangle$ be the corresponding tuple on the H_1-list. Prior to issuing the query, the bit c_i is independent of $\mathcal{A}$'s view — the only value that could be given to $\mathcal{A}$ that depends on c_i is $H(W_i)$, but the distribution on $H(W_i)$ is the same whether $c_i = 0$ or $c_i = 1$. Therefore, the probability that this query causes $\mathcal{B}$ to abort is at most $1/(q_T + 1)$. Since $\mathcal{A}$ makes at most q_T trapdoor queries the probability that $\mathcal{B}$ does not abort as a result of all trapdoor queries is at least $(1 - 1/(q_T + 1))^{q_T} \geq 1/e$. □

Claim 2: The probability that algorithm $\mathcal{B}$ does not abort during the challenge phase is at least $1/q_T$. Hence, $\Pr[\mathcal{E}_2] \geq 1/q_T$.

Proof. Algorithm $\mathcal{B}$ will abort during the challenge phase if $\mathcal{A}$ is able to produce W_0, W_1 with the following property: $c_0 = c_1 = 1$ where for $i = 0, 1$ the tuple $\langle W_i, h_i, a_i, c_i \rangle$ is the tuple on the H_1-list corresponding to W_i. Since $\mathcal{A}$ has not queried for the trapdoor for W_0, W_1 we have that both c_0, c_1 are independent of $\mathcal{A}$'s current view. Therefore, since $\Pr[c_i = 0] = 1/(q_T+1)$ for $i = 0, 1$, and the two values are independent of one another, we have that $\Pr[c_0 = c_1 = 1] = (1 - 1/(q_T + 1))^2 \leq 1 - 1/q_T$. Hence, the probability that $\mathcal{B}$ does not abort is at least $1/q_T$. □

Observe that since $\mathcal{A}$ can never issue a trapdoor query for the challenge keywords W_0, W_1 the two events $\mathcal{E}_1$ and $\mathcal{E}_2$ are independent. Therefore, $\Pr[\mathcal{E}_1 \wedge \mathcal{E}_2] \geq 1/(eq_T)$.

To complete the proof of Theorem 3.1 it remains to show that $\mathcal{B}$ outputs the solution to the given BDH instance with probability at least ϵ/q_{H_2}. To do we show that during the simulation $\mathcal{A}$ issues a query for $H_2(e(H_1(W_b), u_1^{\gamma}))$ with probability at least ϵ.

Claim 3: Suppose that in a real attack game $\mathcal{A}$ is given the public key $[g, u_1]$ and $\mathcal{A}$ asks to be challenged on words W_0 and W_1. In response, $\mathcal{A}$ is given a challenge $C = [g^r, J]$. Then, in the real attack game $\mathcal{A}$ issues an H_2 query for either $H_2(e(H_1(W_0), u_1^r))$ or $H_2(e(H_1(W_1), u_1^r))$ with probability at least 2ϵ.

Proof. Let $\mathcal{E}_3$ be the event that in the real attack $\mathcal{A}$ does not issue a query for either one of $H_2(e(H_1(W_0), u_1^r))$ and $H_2(e(H_1(W_1), u_1^r))$. Then, when $\mathcal{E}_3$ occurs we know that the bit $b \in \{0,1\}$

indicating whether C is a PEKS of W_0 or W_1 is independent of $\mathcal{A}$'s view. Therefore, $\mathcal{A}$'s output b' will satisfy $b = b'$ with probability at most $\frac{1}{2}$. By definition of $\mathcal{A}$, we know that in the real attack $|\Pr[b = b'] - 1/2| \geq \epsilon$. We show that these two facts imply that $\Pr[\neg\mathcal{E}_3] \geq 2\epsilon$. To do so, we first derive simple upper and lower bounds on $\Pr[b = b']$:

$$\begin{aligned}
\Pr[b = b'] &= \Pr[b = b'|\mathcal{E}_3]\Pr[\mathcal{E}_3] + \Pr[b = b'|\neg\mathcal{E}_3]\Pr[\neg\mathcal{E}_3] \\
&\leq \Pr[b = b'|\mathcal{E}_3]\Pr[\mathcal{E}_3] + \Pr[\neg\mathcal{E}_3] = \frac{1}{2}\Pr[\mathcal{E}_3] + \Pr[\neg\mathcal{E}_3] \\
&= \frac{1}{2} + \frac{1}{2}\Pr[\neg\mathcal{E}_3], \\
\Pr[b = b'] &\geq \Pr[b = b'|\mathcal{E}_3]\Pr[\mathcal{E}_3] = \frac{1}{2}\Pr[\mathcal{E}_3] = \frac{1}{2} - \frac{1}{2}\Pr[\neg\mathcal{E}_3].
\end{aligned}$$

It follows that $\epsilon \leq |\Pr[b = b'] - 1/2| \leq \frac{1}{2}\Pr[\neg\mathcal{E}_3]$. Therefore, in the real attack, $\Pr[\neg\mathcal{E}_3] \geq 2\epsilon$ as required. □

Now, assuming $\mathcal{B}$ does not abort, we know that $\mathcal{B}$ simulates a real attack game perfectly up to the moment when $\mathcal{A}$ issues a query for either $H_2(e(H_1(W_0), u_1^\gamma))$ or $H_2(e(H_1(W_1), u_1^\gamma))$. Therefore, by Claim 3, by the end of the simulation $\mathcal{A}$ will have issued a query for either $H_2(e(H_1(W_0), u_1^\gamma))$ or $H_2(e(H_1(W_1), u_1^\gamma))$ with probability at least 2ϵ. It follows that $\mathcal{A}$ issues a query for $H_2(e(H_1(W_b), u_1^\gamma))$ with probability at least ϵ. Consequently, the value $e(H_1(W_b), u_1^\gamma) = e(g^{\beta+a_b}, g)^{\alpha\gamma}$ will appear on the left hand side of some pair in the H_2-list. Algorithm $\mathcal{B}$ will choose the correct pair with probability at least $1/q_{H_2}$ and therefore, assuming $\mathcal{B}$ does not abort during the simulation, it will produce the correct answer with probability at least ϵ/q_{H_2}. Since $\mathcal{B}$ does not abort with probability at least $1/(eq_T)$ we see that $\mathcal{B}$'s success probability overall is at least $\epsilon/(eq_T q_{H_2})$ as required. □

3.2 A limited construction using any trapdoor permutation

Our second PEKS construction is based on general trapdoor permutations, assuming that the total number of keywords that the user wishes to search for is bounded by some polynomial function in the security parameter. (As a first step in our construction, we will make an even stronger assumption that the total number of words $\Sigma \subset \{0,1\}^*$ in the dictionary is also bounded by a polynomial function, we will later show how to remove this additional assumption.) We will also need a family of semantically-secure encryptions where given a ciphertext it is computationally hard to say which public-key this ciphertext is associated with. This notion was formalized by Bellare et al. [1]. We say that a public-key system that has this property is **source-indistinguishable**. More precisely, source-indistinguishability for an encryption scheme (G, E, D) is defined using the following game between a challenger and an attacker $\mathcal{A}$ (here G is the key generation algorithm, and E/D are encryption/decryption algorithms). The security parameter s is given to both players. Source Indistinguishability security game:

1. The challenger runs algorithm $G(s)$ two times to generate two public/private key pairs $(PK_0, Priv_0)$ and $(PK_1, Priv_1)$.
2. The challenger picks a random $M \in \{0,1\}^s$ and a random $b \in \{0,1\}$ and computes an encryption $C = PK_b(M)$. The challenger gives (M, C) to the attacker.
3. The attacker outputs b' and wins the game if $b = b'$.

In other words, the attacker wins if he correctly guesses whether he was given the encryption of M under PK_0 or under PK_1. We define $\mathcal{A}$'s advantage in winning the game as:

$$\mathrm{AdvSI}_{\mathcal{A}}(s) = |\Pr[b = b'] - \frac{1}{2}|$$

Definition 3.2. We say that a public-key encryption scheme is source indistinguishable if for any polynomial time attacker $\mathcal{A}$ we have that $\text{AdvSI}_{\mathcal{A}}(s)$ is a negligible function.

We note that Bellare et al. [1] define a stronger notion of source indistinguishability than the one above by allowing the adversary to choose the challenge message M. For our purposes, giving the adversary an encryption of a random message is sufficient.

Source indistinguishability can be attained from any trapdoor permutation family, where for a given security parameter all permutations in the family are defined over the same domain. Such a family can be constructed from any family of trapdoor permutations as described in [1]. Then to encrypt a bit b we pick a random x, and output $[f(x), GL(x) \oplus b]$ where GL is the Goldreich-Levin hard-core bit [19]. We therefore obtain the following lemma:

Lemma 3.3. *Given any trapdoor permutation family we can construct a semantically secure source indistinguishable encryption scheme.*

More constructions are given in [1]. We note that source indistinguishability is an orthogonal property to semantic security. One can build a semantically secure system that is not source indistinguishable (by embedding the public key in every ciphertext). Conversely, one can build a source indistinguishable system that is not semantically secure (by embedding the plaintext in every ciphertext).

A simple PEKS from trapdoor permutations. When the keyword family Σ is of polynomial size (in the security parameter) it is easy to construct searchable encryption from any source-indistinguishable public-key system (G, E, D). We let s be the security parameter for the scheme.

- KeyGen: For each $W \in \Sigma$ run $G(s)$ to generate a new public/private key pair $PK_W/Priv_W$ for the source-indistinguishable encryption scheme. The PEKS public key is $A_{pub} = \{PK_W \mid W \in \Sigma\}$. The private key is $A_{priv} = \{Priv_W \mid W \in \Sigma\}$.
- PEKS(A_{pub}, W): Pick a random $M \in \{0,1\}^s$ and output PEKS$(A_{pub}, W) = (M,\ E[PK_W, M])$, i.e. encrypt M using the public key PK_W.
- Trapdoor(A_{priv}, W): The trapdoor for word W is simply $T_W = Priv_W$.
- Test(A_{pub}, S, T_W): Test if the decryption $D[T_W, S] = 0^s$. Output 'yes' if so and 'no' otherwise.

Note that the dictionary must be of polynomial size (in s) so that the public and private keys are of polynomial size (in s).

This construction gives a semantically secure PEKS as stated in the following simple theorem. Semantically secure PEKS is defined as in Definition 2.2 except that the adversary is not allowed to make chosen keyword queries.

Theorem 3.4. *The* PEKS *scheme above is semantically secure assuming the underlying public key encryption scheme (G, E, D) is source-indistinguishable.*

Proof sketch: Let $\Sigma = \{W_1, \ldots, W_k\}$ be the keyword dictionary. Suppose we have a PEKS attacker $\mathcal{A}$ for which $\text{Adv}_{\mathcal{A}}(s) > \epsilon(s)$. We build an attacker $\mathcal{B}$ that breaks the source indistinguishability of (G, E, D) where $\text{AdvSI}_{\mathcal{B}}(s) > \epsilon(s)/k^2$.

The reduction is immediate: $\mathcal{B}$ is given two public keys PK_0, PK_1 and a pair (M, C) where M is random in $\{0,1\}^s$ and $C = PK_b(M)$ for $b \in \{0,1\}$. Algorithm $\mathcal{B}$ generates $k-2$ additional public/private keys using $G(s)$. It creates A_{pub} as a list of all k public keys with PK_0, PK_1 embedded in a random location in the list. Let W_i, W_j be the words associated with the public keys PK_0, PK_1. $\mathcal{B}$ sends A_{pub} to $\mathcal{A}$ who then responds with two words $W_k, W_\ell \in \Sigma$ on which $\mathcal{A}$ wishes to be challenged. If $\{i, j\} \neq \{k, \ell\}$ algorithm $\mathcal{B}$ reports failure and aborts. Otherwise, $\mathcal{B}$

sends the challenge (M, C) to $\mathcal{A}$ who then responds with a $b' \in \{0,1\}$. Algorithm $\mathcal{B}$ outputs b' as its response to the source indistinguishability challenge. We have that $b = b'$ if algorithm $\mathcal{B}$ did not abort and $\mathcal{A}$'s response was correct. This happens with probability at least $\frac{1}{2} + \epsilon/k^2$. Hence, $\text{AdvSI}_{\mathcal{B}}(s) > \epsilon(s)/k^2$ as required. □

We note that this PEKS can be viewed as derived from an IBE system with a limited number of identities. For each identity there is a pre-specified public key. Such an IBE system is implied in the work of Dodis et al. [13]. They propose reducing the size of the public-key using cover-free set systems. We apply the same idea below to reduce the size of the public key in the PEKS above.

Reducing the public key size. The drawback of the above scheme is that the public key length grows linearly with the total dictionary size. If we have an upper-bound on the total number of keyword trapdoors that the user will release to the email gateway (though we do not need to know these keywords a-priori) we can do much better using cover-free families [15] and can allow keyword dictionary to be of exponential size. Since typically a user will only allow a third party (such as e-mail server) to search for a limited number of keywords so that assuming an upper bound on the number of released trapdoors is within reason. We begin by recalling the definition of cover-free families.

Definition 3.5. Cover-free families. *Let d, t, k be positive integers, let G be a ground set of size d, and let $F = \{S_1, \ldots, S_k\}$ be a family of subsets of G. We say that subset S_j does not* cover *S_i if it holds that $S_i \not\subseteq S_j$. We say that family F is* t-cover free *over G if each subset in F is not covered by the union of t subsets in F. Moreover, we say that a family of subsets is* q-uniform *if all subsets in the family have size q.*

We will use the following fact from [14].

Lemma 3.6. [14] There exists a deterministic algorithm that, for any fixed t, k, constructs a q-uniform t-cover free family F over a ground set of size d, for $q = \lceil d/4t \rceil$ and $d \leq 16t^2(1 + \log(k/2)/\log 3)$.

The PEKS. Given the previous PEKS construction as a starting point, we can significantly reduce the size of public file A_{pub} by allowing user to re-use individual public keys for different keywords. We associate to each keyword a subset of public keys chosen from a cover free family. Let k be the size of the dictionary $\Sigma = \{W_1, \ldots, W_k\}$ and let t be an upper bound on the number of keyword trapdoors released to the mail gateway by user Alice. Let d, q satisfy the bounds of Lemma 3.6. The $\mathsf{PEKS}(d, t, k, q)$ construction is as follows:

- KeyGen: For $i = 1, \ldots, d$ run algorithm $G(s)$ to generate a new public/private key pair $PK_i/Priv_i$ for the source-indistinguishable encryption scheme. The PEKS public key is $A_{pub} = \{PK_1, \ldots, PK_d\}$. The private key is $A_{priv} = \{Priv_1, \ldots, Priv_d\}$. We will be using a q-uniform t-cover free family of subsets $F = \{S_1, \ldots, S_k\}$ of $\{PK_1, \ldots, PK_d\}$. Hence, each S_i is a subset of public keys.
- $\mathsf{PEKS}(A_{pub}, W_i)$: Let $S_i \in F$ be the subset associated with the word $W_i \in \Sigma$. Let $S_i = \{PK^{(1)}, \ldots, PK^{(q)}\}$. Pick random messages $M_1, \ldots, M_q \in \{0,1\}^s$ and let $M = M_1 \oplus \cdots \oplus M_q$. Output the tuple:

$$\mathsf{PEKS}(A_{pub}, W_i) = \Big(M,\ E[PK^{(1)}, M_1],\ \ldots,\ E[PK^{(q)}, M_q]\Big)$$

- $\mathsf{Trapdoor}(A_{priv}, W_i)$: Let $S_i \in F$ be the subset associated with word $W_i \in \Sigma$. The trapdoor for word W_i is simply the set of private keys that correspond to the public keys in the set S_i.

- $\mathsf{Test}(A_{pub}, R, T_W)$:
  Let $T_W = \{Priv^{(1)}, \ldots, Priv^{(q)}\}$ and let $R = (M,\ C_1,\ \ldots,\ C_q)$ be a PEKS. For $i = 1, \ldots, q$ decrypt each C_i using private key $Priv^{(i)}$ to obtain M_i. Output 'yes' if $M = M_1 \oplus \cdots \oplus M_q$, and output 'no' otherwise.

The size of the public key file A_{pub} is much smaller now: logarithmic in the size of the dictionary. The downside is that Alice can only release t keywords to the email gateway. Once t trapdoors are released privacy is no longer guaranteed. Also, notice that the size of the PEKS is larger now (logarithmic in the dictionary size and linear in t). The following corollary of Theorem 3.4 shows that the resulting PEKS is secure.

Corollary 3.7. *Let d, t, k, q satisfy the bounds of Lemma 3.6. The $\mathsf{PEKS}(d, t, k, q)$ scheme above is semantically secure under a chosen keyword attack assuming the underlying public key encryption scheme (G, E, D) is source-indistinguishable and semantically secure, and that the adversary makes no more than t trapdoors queries.*

Proof sketch: Let $\Sigma = \{W_1, \ldots, W_k\}$ be the keyword dictionary. Suppose we have a PEKS attacker $\mathcal{A}$ for which $\mathrm{Adv}_{\mathcal{A}}(s) > \epsilon(s)$. We build an attacker $\mathcal{B}$ that breaks the source indistinguishability of (G, E, D).

Algorithm $\mathcal{B}$ is given two public keys PK_0, PK_1 and a pair (M, C) where M is random in $\{0,1\}^s$ and $C = PK_b(M)$ for $b \in \{0, 1\}$. Its goal is to output a guess for b which it does by interacting with $\mathcal{A}$. Algorithm $\mathcal{B}$ generates $d - 2$ additional public/private keys using $G(s)$. It creates A_{pub} as a list of all d public keys with PK_0, PK_1 embedded in a random location in the list. Let W_i, W_j be the words associated with the public keys PK_0, PK_1.

$\mathcal{B}$ sends A_{pub} to $\mathcal{A}$. Algorithm $\mathcal{A}$ issues up to t trapdoor queries. $\mathcal{B}$ responds to a trapdoor query for $W \in \Sigma$ as follows: let $S \in F$ be the subset corresponding to the word W. If $PK_0 \in S$ or $PK_1 \in S$ algorithm $\mathcal{B}$ reports failure and aborts. Otherwise, $\mathcal{B}$ gives $\mathcal{A}$ the set of private keys $\{Priv_i \,|\, i \in S\}$.

At some point, Algorithm $\mathcal{A}$ outputs two words $W_0', W_1' \in \Sigma$ on which it wishes to be challenged. Let $S_0', S_1' \in F$ be the subsets corresponding to W_0', W_1' respectively. Let $\mathcal{E}$ be the event that $PK_0 \in S_0'$ and $PK_1 \in S_1'$. If event $\mathcal{E}$ did not happen then $\mathcal{B}$ reports failure and aborts.

We now know that $PK_0 \in S_0'$ and $PK_1 \in S_1'$. For $j = 0, 1$ let $S_j' = \{PK_j^{(1)}, \ldots, PK_j^{(q)}\}$. We arrange things so that $PK_0 = PK_0^{(c)}$ and $PK_1 = PK_1^{(c)}$ for some random $1 \le c \le q$. Next, $\mathcal{B}$ picks random $M_1, \ldots, M_{c-1}, M_{c+1}, \ldots, M_q \in \{0,1\}^s$ and sets $M_c = M$. Let $M' = M_1 \oplus \cdots \oplus M_q$. Algorithm $\mathcal{B}$ defines the following hybrid tuple:

$$R \;=\; \Big(M',\ E[PK_0^{(1)}, M_1],\ \ldots,\ E[PK_0^{(c-1)}, M_{c-1}],\ C,\ E[PK_1^{(c+1)}, M_{c+1}],\ \ldots,\ E[PK_1^{(q)}, M_q]\Big)$$

It gives R as the challenge PEKS to algorithm $\mathcal{A}$. Algorithm $\mathcal{A}$ eventually responds with some $b' \in \{0, 1\}$ indicating whether R is $\mathsf{PEKS}(A_{pub}, W_0')$ or $\mathsf{PEKS}(A_{pub}, W_1')$. Algorithm $\mathcal{B}$ outputs b' as its guess for b. One can show using a standard hybrid argument that if $\mathcal{B}$ does not abort then $|\Pr[b = b'] - \frac{1}{2}| > \epsilon/q^2$. The probability that $\mathcal{B}$ does not abort at a result of a trapdoor query is at least $1 - (tq/d)$. The probability that $\mathcal{B}$ does not abort as a result of the choice of words W_0', W_1' is at least $(q/d)^2$. Hence, $\mathcal{B}$ does not abort with probability at least $1/poly(t, q, d)$. Repeatedly running $\mathcal{B}$ until it does not abort shows that we can get advantage ϵ/q^2 in breaking the source indistinguishability of (G, E, D) in expected polynomial time in the running time of $\mathcal{A}$. □

4 Construction using Jacobi symbols

Given the relation between Identity Based Encryption and PEKS it is tempting to construct a PEKS from an IBE system due to Cocks [3]. The security of Cocks' IBE system is based on the difficulty of distinguishing quadratic residues from non-residues modulo $N = pq$ where $p = q = 3 (\mathrm{mod} 4)$.

Unfortunately, Galbraith [11] shows that the Cocks system as described in [3] is not public-key private in the sense of Bellare et al. [1]. Therefore it appears that the Cocks system cannot be directly used to construct a PEKS. It provides a good example that constructing a PEKS is a harder problem than constructing an IBE.

We briefly explain why the basic Cocks system is not public key private. Let $N \in \mathbb{Z}$ be a product of two primes. For a public key $x \in \mathbb{Z}_N$ and a bit $b \in \{0, 1\}$ define $P_{x,b}$ to be the set

$$P_{x,b} = \{\ r + \frac{x}{r} \quad | \quad r \in \mathbb{Z}_N \ \text{ and } \ (r/N) = (-1)^b \} \ \subseteq \ \mathbb{Z}_N$$

where (r/N) denotes the Jacobi symbol of r over N. Fix some $b \in \{0, 1\}$. To show that the system is not public key private it suffices to show an algorithm that given two distinct public keys x and y in $\mathbb{Z}_N$ can distinguish the uniform distribution on $P_{x,b}$ from the uniform distribution on $P_{y,b}$. The algorithm works as follows (it is given as input $x, y \in \mathbb{Z}_N$ and $z \in \mathbb{Z}_N$ where z is either sampled from $P_{x,b}$ or from $P_{y,b}$):

1. Compute $t = z^2 - 4x \in \mathbb{Z}_N$.
2. If the Jacobi symbol $(t/N) = 1$ output "$z \in P_{x,b}$". Otherwise output "$z \in P_{y,b}$".

When z is sampled from $P_{x,b}$ then $t = z^2 - 4x = (r - (x/r))^2$ and hence t will always have Jacobi symbol 1 over N. When z is sampled from $P_{y,b}$ then we expect t to have Jacobi symbol 1 over N with probability about $1/2$. Therefore, the algorithm has advantage $1/2$ in correctly identifying where z is sampled from. Since a Cocks ciphertext contains many such samples, the algorithm above determines whether a ciphertext is encrypted under the public key x or the public key y with overwhelming probability. Note there is no need to know the plaintext b.

5 Conclusions

We defined the concept of a public key encryption with keyword search (PEKS) and gave two constructions. Constructing a PEKS is related to Identity Based Encryption (IBE), though PEKS seems to be harder to construct. We showed that PEKS implies Identity Based Encryption, but the converse is currently an open problem. Our constructions for PEKS are based on recent IBE constructions. We are able to prove security by exploiting extra properties of these schemes.

Acknowledgments

We thank Glenn Durfee for suggesting the use of H_2 in the construction of Section 3.1. We thank Yevgeniy Dodis, David Molnar, and Steven Galbraith for helpful comments on this work.

References

[1] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval "Key-Privacy in Public-Key Encryption," in Advances in Cryptology - Asiacrypt 2001 Proceedings, LNCS Vol. 2248, Springer-Verlag, 2001.

[2] D. Boneh and M. Franklin, *Identity-based Encryption from the Weil Pairing,* SIAM J. of Computing, Vol. 32, No. 3, pp. 586-615, 2003, Extended abstract in Crypto 2001.

[3] C. Cocks, *An identity based encryption scheme based on quadratic residues*, Eighth IMA International Conference on Cryptography and Coding, Dec. 2001, Royal Agricultural College, Cirencester, UK.

[4] C. Cachin, S. Micali, M. Stadler *Computationally Private Information Retrieval with Polylogarithmic Communication* Eurcrypt 1999.

[5] B. Chor, O. Goldreich, E. Kushilevitz and M. Sudan, *Private Information Retrieval,* in FOCS 95 (also Journal of ACM).

[6] J. Coron, "On the exact security of Full-Domain-Hash", in *Advances in Cryptology – Crypto 2000*, Lecture Notes in Computer Science, Vol. 1880, Springer-Verlag, pp. 229–235, 2000.

[7] D. Dolev, C. Dwork, and M. Naor, "Non-Malleable Cryptography," in SIAM Journal on Computing, 2000. Early version in proceedings of STOC '91.

[8] G. Di Crescenzo, Y. Ishai, and R. Ostrovsky. Universal service-providers for database private information retrieval. In *Proc. of the 17th Annu. ACM Symp. on Principles of Distributed Computing*, pages 91-100, 1998.

[9] G. Di Crescenzo, T. Malkin, and R. Ostrovsky. Single-database private information retrieval implies oblivious transfer. In *Advances in Cryptology - EUROCRYPT 2000*, 2000.

[10] A. Iliev, S. Smith Privacy-enhanced credential services. Second annual PKI workshop. (see also Darthmoth Technical Report TR-2003-442; http://www.cs.dartmouth.edu/ sws/papers/ilsm03.pdf.

[11] S. Galbraith, private communications.

[12] Y. Desmedt, "Computer security by redefining what a computer is," in Proceedings New Security Paradigms II Workshop, pp. 160–166, 1992.

[13] Y. Dodis, J. Katz, S. Xu, and M. Yung. "Key-insulated public key cryptosystems," in Advances in Cryptology – Eurocrypt 2002, LNCS, Springer-Verlag, pp. 65–82, 2002.

[14] D. Z. Du and F. K. Hwang, *Combinatorial Group Testing and its Applications,* World Scientific, Singapore, 1993.

[15] P. Erdos, P. Frankl and Z. Furedi, *Families of finite sets in which no set is covered by the union of r others,* in Israeli Journal of Mathematics, 51: 79–89, 1985.

[16] E. Goh, "Building Secure Indexes for Searching Efficiently on Encrypted Compressed Data," `http://eprint.iacr.org/2003/216/`

[17] O. Goldreich and R. Ostrovsky. Software protection and simulation by oblivious RAMs. *JACM*, 1996.

[18] Goldreich, O., S. Goldwasser, and S. Micali, "How To Construct Random Functions," *Journal of the Association for Computing Machinery*, Vol. 33, No. 4 (October 1986), 792-807.

[19] S. Goldwasser and S. Micali, *Probabilistic Encryption*, in Journal of Computer and System Sciences. vol. 28 (1984), n. 2, pp. 270–299.

[20] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin Protecting data privacy in private information retrieval schemes. In Proc. of the *30th Annual ACM Symposium on the Theory of Computing*, pp. 151-160, 1998.

[21] A. Joux, "The Weil and Tate Pairings as Building Blocks for Public Key Cryptosystems", in *Proc. Fifth Algorithmic Number Theory Symposium*, Lecture Notes in Computer Science, Springer-Verlag, 2002.

[22] E. Kushilevitz and R. Ostrovsky, *Replication is not needed: Single Database, Computationally-Private Information Retrieval,* in FOCS 97.

[23] E. Kushilevitz and R. Ostrovsky. One-way Trapdoor Permutations are Sufficient for Non-Trivial Single-Database Computationally-Private Information Retrieval. In *Proc. of EUROCRYPT '00*, 2000.

[24] P. Maniatis, M. Roussopoulos, E. Swierk, K. Lai, G. Appenzeller, X. Zhao, and M. Baker, *The Mobile People Architecture*. ACM Mobile Computing and Communications Review (MC2R), Volume 3, Number 3, July 1999.

[25] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. In *Proc. of the 31th Annu. ACM Symp. on the Theory of Computing*, pages 245–254, 1999.

[26] R. Ostrovsky. Software protection and simulation on oblivious RAMs. MIT Ph.D. Thesis, 1992. Preliminary version in *Proc. 22nd Annual ACM Symp. Theory Comp.*, 1990.

[27] W. Ogata and K. Kurosawa, "Oblivious keyword search," to appear in J. of Complexity.

[28] D. Song, D. Wagner, and A. Perrig, *Practical Techniques for Searches on Encrypted Data*, in Proc. of the 2000 IEEE symposium on Security and Privacy (S&P 2000).

[29] A. Shamir, *Identity-based Cryptosystems and Signature Schemes,* in CRYPTO 84.

[30] B. Waters, D. Balfanz, G. Durfee, D. Smetters, "Building an encrypted and searchable audit log", to appear in NDSS '04.

A Secure Signature Scheme from Bilinear Maps

Dan Boneh Ilya Mironov Victor Shoup

Computer Science Department
Stanford University
{dabo,mironov}@cs.stanford.edu

Courant Institute
New York University
shoup@cs.nyu.edu

Abstract. We present a new class of signature schemes based on properties of certain bilinear algebraic maps. These signatures are secure against existential forgery under a chosen message attack in the standard model (without using the random oracle model). Security is based on the computational Diffie-Hellman problem. The concrete schemes that we get are the most efficient provable discrete-log type signature schemes to date.

1 Introduction

Provably secure signature schemes can be constructed from the most basic cryptographic primitive, one-way functions [NY89,Rom90]. As is often the case with cryptographic schemes designed from elementary blocks, this signature scheme is somewhat impractical. Over the years several signature schemes were proposed based on stronger complexity assumptions. The most efficient schemes provably secure in the standard model are based on the Strong RSA assumption [GHR99,CS99].

Surprisingly, no scheme based on any discrete logarithm problem comes close to the efficiency of the RSA-based schemes. We give a partial solution to this open problem using bilinear maps. A bilinear map is a function $e\colon \mathbb{G}_0 \times \mathbb{G}_1 \mapsto \mathbb{G}_2$ that is consistent with group operations in both of its arguments, as described in the next section. Our construction gives an existentially unforgeable signature whenever the Computational Diffie-Hellman (CDH) assumption holds in $\mathbb{G}_0 \times \mathbb{G}_1$, that is, no efficient algorithm can compute $g^\alpha \in \mathbb{G}_1$ given the three values $h, h^\alpha \in \mathbb{G}_0$, and $g \in \mathbb{G}_1$. Precise definitions are given in the next section.

Our signature scheme is based on a signature authentication tree with a large branching factor. At a high level, our construction bares some resemblance to the signature schemes of Dwork-Naor [DN94] and Cramer-Damgård [CD96]. Both these schemes are based on the hardness of factoring whereas our scheme is based on CDH.

We obtain a concrete signature scheme by instantiating the bilinear map with the modified Weil or Tate pairings. This is the only known provably secure signature scheme based on the CDH assumption that is more efficient than

the most general constructions of [CD95]. It is interesting to note that recently, Lysyanskaya [Lys02] constructed a verifiable unpredictable function (VUF) secure under the Generalized Diffie-Hellman assumption in the standard model. Such functions (defined in [MRV99]) provide a special type of secure signatures called unique signatures. This construction gives an excellent VUF, but as a signature scheme it compares poorly with the construction of [CD95]. We review other known signature schemes in Section 4.

Bilinear maps such as the Weil or Tate pairing have recently been used to construct a number of new cryptosystems, including three-party key exchange [Jou00], identity based encryption [BF01], short signatures [BLS01], credential systems [Ver01], hierarchical identity based encryption [HL02,GS02], and others. In this paper we show how bilinear maps can be used to construct efficient signature schemes secure in the standard model.

Efficient discrete log based signature schemes are known to exist in the random oracle model [PS96,BLS01]. Security in the random oracle model does not imply security in the real world. In this paper we only study signature schemes secure in the standard complexity model.

2 Mappings with Algebraic Properties

We consider binary maps between groups that are consistent with the group structure of their arguments. Such binary maps are called bilinear. Their formal definition follow.

Definition 1 (Bilinear map). *A function* $e\colon \mathbb{G}_0 \times \mathbb{G}_1 \mapsto \mathbb{G}_2$ *is* bilinear *if for any four elements* $g_1, g_2 \in \mathbb{G}_0$, $H_1, H_2 \in \mathbb{G}_1$ *the following holds:*

$$e(g_1 \circ g_2, H_1) = e(g_1, H_1) \circ e(g_2, H_1) \quad \textit{and} \quad e(g_1, H_1 \circ H_2) = e(g_1, H_1) \circ e(g_1, H_2).$$

In this paper we intentionally limit our scope to finite cyclic groups, which allows us to give more efficient constructions.

Throughout the paper we use the following notation. Small Roman letters $f, g, h, \ldots$ from the lower part of the alphabet denote elements of the group $\mathbb{G}_0$; capital Roman letters $F, G, H, \ldots$ stand for elements of $\mathbb{G}_1$; elements of $\mathbb{G}_2$ are denoted by letters from the end of the alphabet x, y, z.

Our constructions are based on the Computational Diffie Problem (CDH) defined below. However to simplify the exposition we define the notion of a secure bilinear map. We then show that this notion is equivalent to CDH. Informally, we say that a bilinear map $e\colon \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_2$ is secure if, given $g \in \mathbb{G}_0$ and $G, H \in \mathbb{G}_1$, it is hard to find $h \in \mathbb{G}_0$ such that $e(h, H) = e(g, G)$. More precisely, we define secure bilinear maps as follows.

Definition 2 (Secure bilinear map). *A bilinear map* $e\colon \mathbb{G}_0 \times \mathbb{G}_1 \mapsto \mathbb{G}_2$ *is* (t, ε)-secure *if for all* t*-time adversaries* $\mathcal{A}$

$$\mathsf{AdvBLM}_{\mathcal{A}} = \Pr\left[e\Big(\mathcal{A}(g, G, H),\ H \Big) = e(g, G) \;\middle|\; g \stackrel{R}{\leftarrow} \mathbb{G}_0; G, H \stackrel{R}{\leftarrow} \mathbb{G}_1 \right] < \varepsilon.$$

The probability is taken over the coin tosses of the algorithm $\mathcal{A}$ and random choice of g, G, H.

We given two simple examples of bilinear maps that are believed to be secure.

- $e(\cdot,\cdot)\colon \mathbb{G}_0 \times \mathbb{G}_1 \mapsto \mathbb{F}^*_{p^r}$ is the Weil or Tate pairings on an elliptic curve $E/\mathbb{F}_p$ and $\mathbb{G}_0, \mathbb{G}_1$ are distinct subgroups of $E[q]$ for some prime q (recall that $E[q]$ is the subgroup containing all point of order dividing q on some elliptic curve $E/\mathbb{F}_p$). For certain curves $E/\mathbb{F}_p$ one can slightly modify the Weil pairing (as in [BF01]) so that we may take $\mathbb{G}_0 = \mathbb{G}_1$. At any rate, the security of the bilinear map $e(\cdot,\cdot)$ is equivalent to the Computational Diffie-Hellman assumption (CDH) in $\mathbb{G}_0 \times \mathbb{G}_1$. Informally, the CDH assumption in $\mathbb{G}_0 \times \mathbb{G}_1$ means that:

  It is infeasible to find G^a given random group elements $h, h^a \in \mathbb{G}_0$, and $G \in \mathbb{G}_1$. When $\mathbb{G}_0 = \mathbb{G}_1$ this is the standard CDH assumption in $\mathbb{G}_0$.
- Another bilinear map believed to be secure is $r(\cdot,\cdot)\colon \mathbb{Z}^*_N \times \mathbb{Z}^+_{\varphi(N)} \mapsto \mathbb{Z}^*_N$ defined as $r(g, H) = g^H$, where N is a product of two primes. This map is secure under the Strong RSA assumption [BP97]. Briefly, the Strong RSA assumption says that the following problem is difficult to solve:

  For a random $x \in Z^*_N$ find $r > 1$ and $y \in Z^*_N$ such that $y^r = x$.

  We give a short argument why the security of the map $r(\cdot,\cdot)$ is reducible to the Strong RSA assumption. Suppose the map $r(\cdot,\cdot)$ is insecure. Then, given (G, H, g) it is feasible to find an $h \in \mathbb{G}_0$ such that $r(g, G) = r(h, H)$, i.e. $g^G = h^H$. This solution yields (through application of the Extended Euclidean algorithm) a $z \in \mathbb{Z}^*_N$ satisfying $z^a = g$, where $a = H/\gcd(G, H)$. For random $G, H \xleftarrow{R} \mathbb{Z}^+_{\varphi(N)}$ the probability that $a = 1$ is negligible. Therefore breaking the security of the bilinear map r amounts to breaking the Strong RSA assumption. It is not known whether the converse is true.

Next, we show that for finite order groups a bilinear map $e\colon \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_2$ is secure if and only if the CDH assumption holds in $\mathbb{G}_0 \times G_1$. We first precisely define the CDH assumption.

Definition 3 (Computational Diffie-Hellman problem). *The* Computational Diffie-Hellman problem *is (t, ε)-hard if for all t-time adversaries $\mathcal{A}$ we have*

$$\mathsf{AdvCDH}_{\mathcal{A}} = \Pr\left[\mathcal{A}(g, H, H^a) = g^a \;\middle|\; g \xleftarrow{R} \mathbb{G}_0; H \xleftarrow{R} \mathbb{G}_1; a \xleftarrow{R} \mathbb{Z}_{|\mathbb{G}_1|}\right] < \varepsilon.$$

Claim. Suppose that $\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_2$ are cyclic groups of prime order p. Suppose the map $e\colon \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_2$ is non-degenerate in the following sense: $e(h, H) \neq 1$ for some $h \in \mathbb{G}_0$ and $H \in \mathbb{G}_1$. Then the map e is (t, ε)-secure if and only if the CDH problem is (t, ε)-hard.

Proof. First, suppose CDH can be solved in time t with probability at least ε. We give an algorithm to show that the map is not (t, ε)-secure. Let $g \in \mathbb{G}_0$ and $G, H \in \mathbb{G}_1$ where both $G, H \neq 1$. We wish to find $h \in \mathbb{G}_0$ such that

$e(g, G) = e(h, H)$. Since $\mathbb{G}_1$ is cyclic of prime order p there exists an $a \in \mathbb{Z}_p$ such that $G = H^a$. Let $h = g^a$. Then h satisfies

$$e(h, H) = e(g^a, H) = e(g, H^a) = e(g, G).$$

Therefore, $h = g^a$, which is the solution to the CDH problem (g, H, G), is the required h. Hence, if the map is (t, ε)-secure then CDH is (t, ε)-hard.

Conversely, suppose there is a t-time algorithm that given random (g, G, H) outputs $h \in \mathbb{G}_0$ such that $e(g, G) = e(h, H)$ with probability at least ε. We show how to solve CDH. Let (g, H, G) be a random instance of the CDH problem, where $H \neq 1$. Write $G = H^a$ for some $a \in \mathbb{Z}_p$. Let h be such that $e(g, G) = e(h, H)$. Then

$$e(h, H) = e(g, G) = e(g, H^a) = e(g^a, H)$$

and hence $e(h/g^a, H) = 1$. Since $H \neq 1$ it follows that $h = g^a$, since otherwise the map e would be degenerate. Hence, if CDH is (t, ε)-hard then the map is (t, ε)-secure. □

3 Security for signature schemes

We recall the standard definition of secure signature schemes stated in terms of exact security, in the spirit of [BR94]. This notion of existential unforgeability under adaptive chosen-message attack is due to [GMR88].

A signature scheme is a triple of probabilistic algorithms: a key generation algorithm KeyGen, signer $\mathsf{Sign}(SK, Msg)$, and verifier $\mathsf{Verify}(Msg, Sig, PK)$. By convention, Verify outputs 1 if it accepts the signature and 0 otherwise. We use the oracle notation, where $A^{B(\cdot)}(\cdot)$ means A supplied with oracle access to B.

Definition 4. *A signature scheme is* (t, ε, n)-existentially unforgeable under adaptive chosen-message attack*, if for all pairs of algorithms* $\mathcal{F}_1, \mathcal{F}_2$ *running in time at most* t

$$\mathsf{AdvSig}_{\mathcal{F}_1, \mathcal{F}_2} = \Pr[\mathsf{Verify}(\mathcal{F}_2(T), PK) = 1 \mid (SK, PK) \leftarrow \mathsf{KeyGen}(1^k);\ T \leftarrow \mathcal{F}_1^{\mathsf{Sign}(SK, \cdot)}(1^k)] < \varepsilon.$$

$\mathcal{F}_1$ *requests no more than* n *signatures from* Sign *and the message output by* $\mathcal{F}_2$ *is different from the messages signed by* Sign*. The probability is taken over the coin tosses of* KeyGen*,* Sign*,* $\mathcal{F}_1$ *and* $\mathcal{F}_2$*. Here* $\mathcal{F}_2(T)$ *outputs a message/signature pair.*

4 Previous work

The seminal paper [GMR84] formulated a strong notion of security for signature schemes: existential unforgeability under adaptive chosen-message attacks. Since

then there have been many proposals for signature schemes meeting this notion of security based on different assumptions and of varying efficiency. Some of these results are summarized in Table 1. With an exception of GMR, all schemes have running time of the signing and verification algorithms proportional to the signature length. This information is omitted from the table.

Reference	**Signature length**	**Public key length**	**Max number of signatures**	**Security assumption**
[GMR84]	$O(k\ell)$	$O(k)$	2^ℓ	claw-free trapdoor permutations
[NY89]	$O(k^2\ell)$	$O(k)$	2^ℓ	UOWHF
[CD95]	$O(k\ell)$	$O(k)$	2^ℓ	one-way homomorphism
[DN94]	$O(k\ell)$	$O(kn)$	n^ℓ	RSA assumption
[CD96]	$O(k\ell)$	$O(k+n)$	n^ℓ	RSA assumption
[GHR99],[CS99]	$O(k)$	$O(k)$	∞	Strong RSA assumption
[Lys02]	$O(km)$	$O(km)$	2^m	Generalized Diffie-Hellman
this paper	$O(k\ell)$	$O(kn)$	n^ℓ	Computational Diffie-Hellman (secure bilinear maps)

Table 1. Summary of provably secure signature schemes. k is the security parameter, ℓ is the depth of the authentication three, n is the branching factor of the tree, and m is the message length. The O-notation refers to asymptotics as a function of the security parameter k.

A signature scheme may be based on the most general cryptographic assumption, namely that one-way functions exist [NY89,Rom90]. The proof proceeds via constructing an authentication tree and results in a scheme in which the signature length is proportional to the binary logarithm of the total number of messages signed with the same public key. More efficient (in terms of the signature length) signature schemes can be based on the Strong RSA assumption or its variants [CS99,GHR99]. Important steps in constructing these schemes were the Dwork-Naor scheme [DN94] later improved by [CD96]. Both schemes use trees with a large branching factor, which are therefore very shallow.

The Dwork-Naor trick is crucial for understanding this paper. In a nutshell, the trick allows to increase the tree's branching factor without blowing up the signature size. An authentication-tree scheme produces signatures that represent paths connecting messages and the root of the tree. Messages are usually placed in the very bottom level of the tree, though [CD95] puts messages in the leaves hanging from the inner nodes. The authentication mechanism works inductively: the root authenticates its children, they authenticate their children, and so on, down to the message authenticated by its parent. If the authentication mechanism allows attaching children to a node only when some secret information is known, then the adversary cannot forge signatures without knowing that secret.

[GMR84] and [CD95] take similar approaches in designing the authentication mechanism (hence the identical asymptotic of their signature length). They concatenate bit representations of a node's children and compute a value that

authenticates this string in respect to the node. To verify a node's authenticity we must know all siblings of the node. If the tree is binary, the signature contains twice as many nodes as the the depth of the tree. Indeed, each node must be accompanied by its sibling.

Since the signature length is proportional to the depth of the tree, one may wonder whether increasing the tree's branching factor is a good idea. The following simple computation shows why this is counterproductive.

Suppose one wants to sign N messages. If the authentication tree is binary, its depth must be at least $\log_2 N$. The signature length is roughly $2k \log_2 N$, where k is the security parameter that accounts for the nodes' size. When the branching factor of the tree is increased from 2 to d, the depth of the tree goes down to $\log_d N$. The signatures, however, must include all siblings of the nodes on the authentication path (ancestors of the message-leaf). Thus the signature size becomes $dk \log_d N$, which is actually *more* than in the binary case.

The improvement achieved by [DN94] is due to the way the sibling nodes are authenticated. Using a stronger complexity assumption than in the GMR scheme, authenticity of a node can be verified given its parent and its authentication value, whose size is independent of the branching factor. Each node has the authentication value different from those of its siblings and their authenticity can be verified independently. It allows to increase the number of a node's children and decrease the depth of the tree without inflating the signature length. The public key size, being the branching factor *times* the security parameter and, because of that, the main drawback of [DN94], was reduced in the Cramer-Damgård scheme [CD96]. Finally, two independent methods proposed in [CS99] and [GHR99] make the tree flat by letting the signer (but not the adversary) add branches on the fly.

We do not distinguish between statefull and stateless schemes. All schemes can be made memoryless at the price of doubling the tree depth [Gol86]. To do so [Gol86] suggested using a pseudo-random function (PRF) to generate all secret values in internal nodes of the authentication tree. The key for the PRF is kept secret at the signer. Furthermore, instead of sequentially stepping through the leaves of the tree (one leaf per signature) we pick a random leaf for every signature. To make sure the same leaf is not chosen at random for two different signatures we need to square the number of leaves and hence double the depth of the tree.

In this paper we implement the Dwork-Naor method using a secure bilinear map. Improving the scheme further in the direction of [CD96,CS99,GHR99] is left as an open problem.

5 The new signature scheme

We present a signature scheme based on a secure bilinear map. An instantiation of this construction yields the most efficient provably secure scheme based on the Computational Diffie-Hellman assumption known to date.

The signature scheme is designed as follows.

– **Setup of the scheme.** Groups $\mathbb{G}_0$ and $\mathbb{G}_1$ have prime order p. Bilinear map $e\colon \mathbb{G}_0 \times \mathbb{G}_1 \mapsto \mathbb{G}_2$ is secure. $\mathcal{H}_k\colon \mathcal{M} \mapsto \{0,1\}^s$ is a family of collision-resistant hash functions, where $\mathcal{M}$ is the message space (the key for $\mathcal{H}_k$ is fixed at random during key generation and made public). The signature scheme allows signing of ℓ^n messages, where ℓ and n are arbitrary positive integers.
– **Key generation.** The public and private keys are created as follows:
  Step 1. Pick $\alpha_1, \ldots, \alpha_n \stackrel{R}{\leftarrow} \mathbb{Z}_p^*$ and $H \stackrel{R}{\leftarrow} \mathbb{G}_1$. Choose a random key k for the collision resistant hash function $\mathcal{H}_k$. Compute $H_1 \leftarrow H^{1/\alpha_1}, \ldots, H_n \leftarrow H^{1/\alpha_n} \in \mathbb{G}_1$.
  Step 2. Pick $g \stackrel{R}{\leftarrow} \mathbb{G}_0$. Compute $y \leftarrow e(g, H)$.
  Step 3. Pick $\beta_0 \stackrel{R}{\leftarrow} \mathbb{Z}_p$. Compute $x_0 \leftarrow e(g, H)^{\beta_0}$.
  Step 4. The **public key** is $k, H, H_1, \ldots, H_n, y$ and x_0. The **private key** is $\alpha_1, \ldots, \alpha_n, \beta_0$, and g.
– **Signing algorithm.** Each node in the tree is authenticated with respect to its parent; messages to be signed are authenticated with respect to the leaves, which are selected in sequential order and never reused. To sign the i^{th} message $M \in \mathcal{M}$ the signer generates the i^{th} leaf of the authentication tree together with a path from the leaf to the root. We denote the leaf by $x_\ell \in \mathbb{G}_1$ and denote the path from the leaf to the root by $(x_\ell, i_\ell, x_{\ell-1}, i_{\ell-1}, \ldots, i_1, x_0)$, where x_j is the i_j^{th} child of x_{j-1} (here $i_j \in \{1, \ldots, n\}$). The leaf and the nodes along the path and their authentication values are computed as follows:
  Step 1. All nodes of the tree, including the leaf x_ℓ, that have not been visited before are computed as $x_j \leftarrow e(g, H)^{\beta_j}$ for some random $\beta_j \stackrel{R}{\leftarrow} \mathbb{Z}_p$. The secret β_j is stored for as long as node x_j is an ancestor of the current signing leaf (β_ℓ is discarded immediately after use). Note that when using stateless signatures [Gol86] the β_j are derived using a PRF and hence there is no need to remember their values.
  Step 2. The authentication value of x_j, the i_j^{th} child of x_{j-1}, is $f_j \leftarrow g^{\alpha_{i_j}(\beta_{j-1} + \mathcal{H}_k(x_j))}$.
  Step 3. The authentication value of $\mathcal{H}_k(M)$, the child of x_ℓ, is $f \leftarrow g^{\beta_\ell + \mathcal{H}_k(M)}$.
  Step 4. The **signature** on M is $(f, f_\ell, i_\ell, \ldots, f_1, i_1)$.
– **Verification algorithm.** To verify a signature $(\hat{f}, \hat{f}_\ell, \hat{i}_\ell, \ldots, \hat{f}_1, \hat{i}_1)$ on a message M we reconstruct the nodes $\hat{x}_\ell, \ldots, \hat{x}_0$ in a bottom-up order (from leaf $\hat{x}_\ell$ to root $\hat{x}_0$). The signature is accepted if and only if $\hat{x}_0$ matches the root of the tree. More precisely, to verify the signature the verifier performs the following steps:
  Step 1. Compute $\hat{x}_\ell \leftarrow e(\hat{f}, H) \cdot y^{-\mathcal{H}_k(M)}$.
  Step 2. For $j = \ell \ldots 1$ compute $\hat{x}_{j-1} \leftarrow e(\hat{f}_j, H_{i_j}) \cdot y^{-\mathcal{H}_k(\hat{x}_j)}$.
  Step 3. The signature is accepted if $\hat{x}_0 = x_0$.

A signature output by the signing algorithm passes the verification test. Indeed, step 1 of the verification algorithm results in

$$e(f, H) \cdot y^{-\mathcal{H}_k(M)} = e(g^{\beta_\ell + \mathcal{H}_k(M)}, H) \cdot e(g, H)^{-\mathcal{H}_k(M)} = e(g^{\beta_\ell}, H) = x_\ell.$$

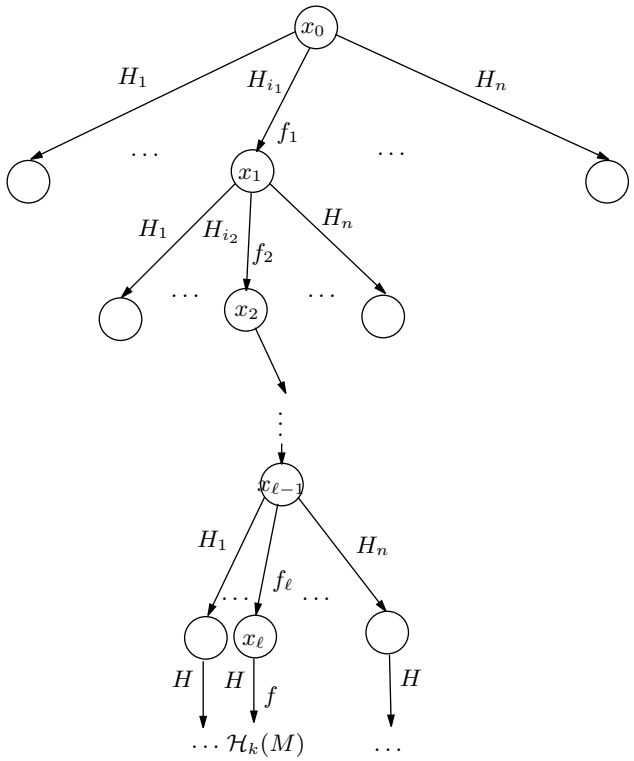

Fig. 1. Authentication tree. The signature on M is $(f, f_\ell, i_\ell, \ldots, f_1, i_1)$.

For any $j \in 1 \ldots \ell$ the result of computation in step 2 of the verification algorithm is

$$e(f_j, H_{i_j}) \cdot y^{-\mathcal{H}_k(x_j)} = e(g^{\alpha_{i_j}(\beta_{j-1}+\mathcal{H}_k(x_j))}, H^{1/\alpha_{i_j}}) \cdot e(g, H)^{-\mathcal{H}_k(x_j)} = \\ e(g^{\beta_{j-1}+\mathcal{H}_k(x_j)}, H) \cdot e(g^{-\mathcal{H}_k(x_j)}, H) = e(g^{\beta_{j-1}}, H) = x_{j-1}.$$

Signature length. Suppose the user needs to generate a billion signatures. Then taking $n = 20$ and $\ell = 4$ is sufficient ($4^{20} > 10^{12}$). The public key will contain 20 elements in $\mathbb{G}_1$ and two elements in $\mathbb{G}_2$. The signature contains five elements in $\mathbb{G}_0$. To get stateless signatures we would need to double the depth so that each signature will contain nine elements of $\mathbb{G}_0$.

Security. We show that the signature scheme is secure against the most general attack. To formalize it as a claim we need to recall the definition of collision-resistance and assume the existence of a collision-finding algorithm for e.

Definition 5 (Collision-resistant function). *We call function* $\mathcal{H}\colon \mathcal{K}\times\mathcal{M}\mapsto\{0,1\}^s$ *a family of* (t,ε)-collision-resistant *functions, if for any* t*-time algorithm* $\mathcal{A}$ *its advantage in finding a collision*

$$\mathsf{AdvCR}_{\mathcal{A}} = \Pr[\mathcal{H}_k(M_1)=\mathcal{H}_k(M_2), M_1\neq M_2 \mid k \xleftarrow{R} \mathcal{K}; (M_1,M_2)\leftarrow\mathcal{A}(k)] < \varepsilon.$$

The probability is taken over $\mathcal{A}$*'s coin tosses.*

Definition 6 (Collision-finding algorithm). *We say that an algorithm is* collision-finding *for bilinear map* $e\colon \mathbb{G}_0\times\mathbb{G}_1\mapsto\mathbb{G}_2$ *if it outputs a solution* $g', g''\in\mathbb{G}_0$*, where* $g', g''\neq 1$*, to the equation*

$$e(g', G') = e(g'', G''),$$

for given $G', G''\in\mathbb{G}_1$ *whenever such a solution exists.*

For the bilinear maps defined on elliptic curves, namely the Weil and Tate pairings, it is easy to build collision finding algorithms. This yields a concrete discrete-log signature scheme as described later in this section.

Theorem 1. *The signature scheme is* $(t,\varepsilon/(n+1),m)$*-secure against existential forgery against adaptive chosen-message attack under the following assumptions:*

- e *is a* (t,ε)*-secure bilinear map;*
- $\mathcal{H}$ *is* (t,ε)*-collision-resistant function;*
- *there is an efficient collision-finding algorithm for* e*.*

Proof. The proof is by contradiction. We show that existence of an efficient forger implies that we can either find a collision in $\mathcal{H}$ or solve

$$e(g^*, G_1) = e(g_2, G_3)$$

for g^* given $G_1, G_3\in\mathbb{G}_1$ and $g_2\in\mathbb{G}_0$.

To this end we set up the public key to help the simulator in answering the adversary's signing queries. In the same time a forgery would let us respond to the challenge or find a collision of the hash function with a non-negligible probability.

First, we set $y\leftarrow e(g_2, G_3)$. Second, we pick random $i\in\{0,\ldots,n\}$. Consider two cases.

Case 1: $i=0$**.** We assign $H\leftarrow G_1$, pick $\gamma_j \xleftarrow{R} \mathbb{Z}_p$ for all $j\in 1\ldots n$ and assign $H_j\leftarrow G_3^{1/\gamma_i}$. All internal nodes of the authentication tree, including the root x_0 but excluding the leaves, are computed as random powers of $e(g_2, G_3)$. Suppose x' is the j^{th} child of x'', where $x''=e(g_2,G_3)^{\gamma}$. The authentication value for x' is computed as $f'\leftarrow g_2^{\gamma_j(\gamma+\mathcal{H}(x'))}$. It correctly authenticates x' as the j^{th} child of x'', since

$$e(f', H_j)\cdot y^{-\mathcal{H}(x')} = e(g_2^{\gamma_j(\gamma+\mathcal{H}(x'))}, G_3^{1/\gamma_j})\cdot e(g_2,G_3)^{-\mathcal{H}(x')} = e(g_2,G_3)^{\gamma} = x''.$$

When the adversary requests a signature on a message M, the path (i_ℓ, $x_{\ell-1}$, $i_{\ell-1}$, …, i_1,x_0) is generated and the authentication values for all internal nodes along the path are included in the signature. The leaf x_ℓ is computed as $x_\ell \leftarrow e(g_2, G_1^\gamma \cdot G_3^{-\mathcal{H}(M)})$ for a randomly chosen $\gamma \stackrel{R}{\leftarrow} \mathbb{Z}_p$. The authentication value for $\mathcal{H}(M)$ is set to be $f \leftarrow g_2^\gamma$. The leaf x_ℓ is authenticated as the i_ℓ^{th} child of $x_{\ell-1}$ as above. This results in a valid signature $(f, f_\ell, i_\ell, \ldots, f_1, i_1)$.

Case 2: $i \in \{1, \ldots, n\}$**.** Assign $H_i \leftarrow G_1$. We randomly choose $\gamma, \gamma_1, \ldots, \gamma_{i-1}, \gamma_{i+1}, \ldots, \gamma_n \stackrel{R}{\leftarrow} \mathbb{Z}_p$, assign $H \leftarrow G_3^{1/\gamma}$ and $H_j \leftarrow G_3^{1/\gamma_j}$, for all $j \neq i$. We apply the collision-finding algorithm that returns d_1 and d_2 satisfying

$$e(d_1, G_1) = e(d_2, G_3).$$

The authentication tree is constructed from the bottom up. Suppose we are given a set of siblings $z_1, \ldots, z_n$. The challenge is to define their parent node z such that we may find authentication values for all of them. Let $z \leftarrow e(d_2, G_3^\delta) y^{-\mathcal{H}(z_i)}$ for a randomly chosen $\delta \stackrel{R}{\leftarrow} \mathbb{Z}_q$. For all $j \in 1 \ldots n$, such that $j \neq i$, the authentication value of z_j can be computed as $f_j \leftarrow (g_2^{\mathcal{H}(z_j)-\mathcal{H}(z_i)} \cdot d_1^\delta)^{\gamma_j}$. Indeed, for all such j

$$\begin{aligned} e(f_j, H_j) \cdot y^{-\mathcal{H}(z_j)} &= e((g_2^{\mathcal{H}(z_j)-\mathcal{H}(z_i)} \cdot d_2^\delta)^{\gamma_j}, G_3^{1/\gamma_j}) \cdot e(g_2, G_3)^{-\mathcal{H}(z_j)} = \\ & e(g_2^{\mathcal{H}(z_j)-\mathcal{H}(z_i)} \cdot d_2^\delta, G_3) \cdot e(g_2^{-\mathcal{H}(z_j)}, G_3) = e(g_2^{-\mathcal{H}(z_j)} \cdot d_2^\delta, G_3) = \\ & e(d_2, G_3^\delta) \cdot e(g_2, G_3)^{-\mathcal{H}(z_i)} = e(d_2, G_3^\delta) y^{-\mathcal{H}(z_i)} = z_j. \end{aligned}$$

Node z_i is authenticated with $f_i \leftarrow d_1^\delta$. Indeed,

$$e(f_i, H_i) \cdot y^{-\mathcal{H}(z_i)} = e(d_1^\delta, G_1) \cdot y^{-\mathcal{H}(z_i)} = e(d_2^\delta, G_3) \cdot y^{-\mathcal{H}(z_i)} = z.$$

It follows that every internal node may be computed given its j^{th} child. In particular, the root of the tree, x_0, is determined by values on the path (x_ℓ, j, $x_{\ell-1}$,…,x_1, j), which can be efficiently computed by the randomized algorithm given above.

When a signature is requested by the adversary, a new leaf is generated as $x_\ell \leftarrow e(g_2, H^\delta)$, where $\delta \stackrel{R}{\leftarrow} \mathbb{Z}_p$. Then the path to the root is computed, which would involve generating new nodes from their j^{th} children. The authentication value for $\mathcal{H}(M)$ is given by $f \leftarrow g_2^{\delta+\gamma\mathcal{H}(M)}$.

We have shown that the simulator may effectively replace the signing oracle and answer the adversary's queries. The simulated answers have the same distribution as signatures output by the real signer.

Solving the challenge. Let us consider an algorithm that interacts with the signing oracle and then forges a signature $(f, f_\ell, i_\ell, \ldots, f_1, i_1)$ on a message M. Without loss of generality we may assume that the adversary makes ℓ^n queries. Denote the full authentication tree constructed by the simulator by T. The signature on M has the form of an authenticated path up the tree from a leaf to the root x_0.

Let $(x_\ell, i_\ell, x_{\ell-1}, i_{\ell-1}, \ldots, i_1, x_0)$ be the path reconstructed from the signature. Define x_j as the lowest node of the path that also appears in T.

We distinguish between two types of forgeries:

Type I: $j = \ell$. Denote the message that was previously authenticated using x_ℓ by M' and let f' be the authentication value of $\mathcal{H}(M')$. It follows that

$$e(f, H) \cdot y^{-\mathcal{H}(M)} = x_\ell,$$
$$e(f', H) \cdot y^{-\mathcal{H}(M')} = x_\ell,$$

from which we have

$$y^{\mathcal{H}(M)-\mathcal{H}(M')} = e(f/f', H). \tag{1}$$

Type II: $j < \ell$. Let the x'_{j+1} be the i_j^{th} child of x_j in T and f'_j be its authentication value. Similarly, there are two equations

$$e(f_j, H_{i_j}) \cdot y^{-\mathcal{H}(x_{j+1})} = x_j,$$
$$e(f'_j, H_{i_j}) \cdot y^{-\mathcal{H}(x'_{j+1})} = x_j,$$

that imply

$$y^{\mathcal{H}(x_{j+1})-\mathcal{H}(x'_{j+1})} = e(f_j/f'_j, H_{i_j}). \tag{2}$$

Consider two possibilities. If $\mathcal{H}(M) = \mathcal{H}(M')$ in type I or $\mathcal{H}(x_{j+1}) = \mathcal{H}(x'_{j+1})$ in type II forgery, then we find a collision in the hash function. Otherwise, we solved equation (1) or (2) of the type

$$y^d = e(\hat{f}, \hat{H}),$$

for d and $\hat{f}$, where $d \neq 0$ and $\hat{H}$ is one of $H, H_1, \ldots, H_n$.

Recall that $y = e(g_2, G_3)$. Since the simulation of the adversary's view is perfect, the probability that $\hat{H} = G_1$ is $\frac{1}{n+1}$. Thus $e(\hat{f}^{1/d}, G_1) = e(g_2, G_3)$ and $\hat{f}^{1/d} = g^*$, i.e. a solution to the challenge.

Therefore an efficient algorithm for forging a signature can be used to either find a collision in the hash function or disprove security of e. This completes the proof of the theorem. □

6 Concrete signature scheme

To obtain a concrete secure signature scheme we instantiate the bilinear map $e\colon \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_2$ with a map for which there is an efficient collision finding algorithm. Let $E/\mathbb{F}_p$ be an elliptic curve. We denote by $E(\mathbb{F}_{p^r})$ the group of points of E over $\mathbb{F}_{p^r}$. Let q be a prime divisor of $|E(\mathbb{F}_p)|$.

When using a supersingular curve over $\mathbb{F}_p, p > 3$ we can take $\mathbb{G}_0 = \mathbb{G}_1$ as the subgroup containing all points of order q in $E(\mathbb{F}_p)$. The group $\mathbb{G}_2$ is the subgroup of order q of $\mathbb{F}^*_{p^2}$. The modified Weil pairing [BF01], denoted $\hat{e}$, is a non-degenerate bilinear map on $\mathbb{G}_0 \times \mathbb{G}_1$. Since the CDH assumption is believed

to hold for the group $\mathbb{G}_0$, we know by Claim 2 that $\hat{e}$ is secure in the sense of Definition 2. Since the modified Weil pairing is symmetric (i.e. $\hat{e}(G,H) = \hat{e}(H,G)$ for all $H, G \in \mathbb{G}_0$) a collision finding algorithm for $\hat{e}$ is immediate: given $G, H \in \mathbb{G}_1$ we output $H, G \in \mathbb{G}_0$ as a collision. Indeed, $\hat{e}(G,H) = \hat{e}(H,G)$ and hence H, G is a collision for G, H. Since $\hat{e}$ is secure and there is an efficient collision finder we obtain the following corollary to Theorem 1:

Corollary 1. *The signature scheme of Section 5 instantiated with the modified Weil pairing $\hat{e}$ is existentially unforgeable under a chosen message attack if the CDH assumption holds for the group $\mathbb{G}_0 = \mathbb{G}_1$ defined above.*

To obtain shorter signatures one can avoid using supersingular curves and instead use a family of curves due to Miyaji et al. [MNT01]. Curves $E/\mathbb{F}_p, p > 3$ in this family are not supersingular and have the property that if q divides $|E(\mathbb{F}_p)|$ then $E[q]$ is contained in $E(\mathbb{F}_{p^6})$. Let $\mathbb{G}_0$ be the subgroup of order q in $E(\mathbb{F}_p)$ and let $\mathbb{G}_1$ be some other subgroup of order q in $E(\mathbb{F}_{p^6})$. The Weil pairing e on $\mathbb{G}_0 \times \mathbb{G}_1$ is not degenerate. Furthermore, since CDH is believed to be hard on $\mathbb{G}_0 \times \mathbb{G}_1$, we know by Claim 2 that e is a secure bilinear map in the sense of Definition 2. To build a collision finder for e we use the trace map $\mathrm{tr}\colon E(\mathbb{F}_{p^6}) \to E(\mathbb{F}_p)$. For almost all choices of $\mathbb{G}_1$ the map tr defines an isomorphism from $\mathbb{G}_1$ to $\mathbb{G}_0$. Then, a collision finder for e works as follows: given $(G,H) \in \mathbb{G}_1$ it outputs as a collision the pair $(\mathrm{tr}(G), \mathrm{tr}(H)) \in \mathbb{G}_0$. Indeed, one can verify that $e(\mathrm{tr}(G), H) = e(\mathrm{tr}(H), G)$. Therefore, by Theorem 1 our signature scheme is secure when using this family of curves.

We note that the RSA function $r(x,d) = x^d \mod N$ while being bilinear does not satisfy the condition of Theorem 1 since the orders of groups $\mathbb{G}_0$ and $\mathbb{G}_1$ are not known to the simulator. Nevertheless, the signature scheme can be instantiated with this function, which would yield a scheme similar to the Dwork-Naor scheme.

7 Conclusion

We presented a new signature scheme secure in the standard model (i.e. without random oracles) using groups in which the Computation Diffie-Hellman assumption holds. Our scheme can be implemented using any secure bilinear map (secure in the sense of Definition 2). Instantiating our signature scheme using the Weil or Tate pairings gives the most efficient known discrete-log type signature secure without random oracles.

References

[BP97] N. Barić and B. Pfitzmann, "Collision-free accumulators and fail-stop signature schemes without trees," Proc. of Eurocrypt'97, pp. 480-494, 1997.

[BF01] D. Boneh and M. Franklin, "Identity based encryption from the Weil pairing," Proc. of CRYPTO'01, pp. 213-229, 2001. Also `http://eprint.iacr.org/2001/090/`

[BLS01] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing," Proc. of Asiacrypt'01, pp. 514-532, 2001.

[BR94] M. Bellare and P. Rogaway, "Optimal asymmetric encryption—how to encrypt with RSA," Proc. of Eurocrypt'94, pp. 92–111, 1994.

[CD95] R. Cramer and I. Damgård, "Secure signature schemes based on interactive protocols," Proc. of CRYPTO'95, pp. 297–310, 1995.

[CD96] R. Cramer and I. Damgård, "New generation of secure and practical RSA-based signatures," Proc. of CRYPTO'96, pp. 173–185, 1996.

[CS99] R. Cramer and V. Shoup, "Signature schemes based on the Strong RSA assumption," Proc. of ACM CCS'99, pp. 46–51, 1999. Full version appeared in ACM TISSEC, v. 3(3), pp. 161–185, 2000.

[DN94] C. Dwork and M. Naor, "An efficient existentially unforgeable signature scheme and its applications," Proc. of CRYPTO'94, pp. 234–246, 1994. Full version appeared in J. of Cryptology, v. 11(2), pp. 187–208, 1998.

[FFS88] U. Feige, A. Fiat, and A. Shamir, "Zero-knowledge proofs of identity," J. of Cryptology, v. 1, pp. 77-94, 1988.

[GHR99] R. Gennaro, S. Halevi, and T. Rabin, "Secure hash-and-sign signatures without the random oracle," Proc. of Eurocrypt'99, pp. 123–139, 1999.

[GS02] C. Gentry and A. Silverberg, "Hierarchical ID-based cryptography", Proc. of Asiacrypt'02, pp. 548–566, 2002.

[Gol86] O. Goldreich, "Two remarks concerning the Goldwasser-Micali-Rivest signature scheme," Proc. of CRYPTO'86, pp. 104–110, 1986.

[GMR84] S. Goldwasser, S. Micali, and R. Rivest, "A 'paradoxical' solution to the signature problem (extended abstract)," Proc. of FOCS'84, pp. 441–448, 1984. Journal version in [GMR88].

[GMR88] S. Goldwasser, S. Micali, and R. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," SIAM J. on Computing, 17(2), pp. 281–308, 1988.

[HL02] J. Horwitz and B. Lynn, "Towards hierarchical identity-based encryption", Proc. of Eurocrypt'02, pp. 466–481, 2002.

[Jou00] A. Joux, "A one-round protocol for tripartite Diffie-Hellman," Proc. of ANTS'00, pp. 385–394, 2000.

[Lys02] A. Lysyanskaya, "Unique signatures and verifiable random functions from DH-DDH separation," Proc. of CRYPTO'02, pp. 597–612, 2002.

[MRV99] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," Proc. of FOCS'99, pp. 120–130, 1999.

[MNT01] A. Miyaji, M. Nakabayashi, and S. Takano, "New explicit condition of elliptic curve trace for FR-reduction," IEICE Trans. Fundamentals, v. E84 A(5), May 2001.

[NY89] M. Naor and M. Yung, "Universal one-way hash functions and their cryptographic applications," Proc. of STOC'89, pp. 33–43, 1989.

[PS96] D. Pointcheval and J. Stern, "Security proofs for signature schemes," in Proc. of Eurocrypt'96, pp. 387–398, 1996.

[Rom90] J. Rompel, "One-way functions are necessary and sufficient for secure signatures," Proc. of STOC'90, pp. 387–394, 1990.

[Ver01] E. Verheul, "Self-Blindable Credential Certificates from the Weil Pairing," Proc. of Asiacrypt'01, pp. 533–551, 2001.